Università degli Studi di Cagliari
Facoltà di Ingegneria
Dipartimento di Ingegneria Elettrica ed Elettronica

The Communicational Side of Open Source Communities

Selene Uras

Supervisor
Prof. Giulio Concas

PhD coordinator
Prof. Alessandro Giua

PhD Thesis
Corso di Dottorato di Ricerca in Ingegneria Elettronica ed Informatica
Scuola di Dottorato in Ingegneria dell'Informazione
ING-INF/05
XX ciclo

February 2008

Chi legge le mie parole sta inventandole

Jorge Luis Borges

Contents

List of Figures

List of Tables

Abstract

What matter most in team software development is communication

Kent Beck

Communication is a fundamental value of software development teams, especially in Open Source (OS) communities. An OS community, in fact, is a quite complex network of individuals having different roles and responsibilities, who can be looked upon as volunteers who spend their time creating and improving software. These people taking part in OS Software development use to share knowledge among themselves, exchange information and create a collaborative environment.

To coordinate and improve communication of these teams dis-located all over the world and used to work at different times and ways, it is necessary to predispose and utilize specific tools.

This research study was born with the proposal to individuate and evaluate communication among members of OS communities analysing different development teams.

The starting point is an academic software development project: Metodologie Agili Distribuite (Agile Distributed Methodologies, MAD) project. This case study, performed at University of Cagliari, was made up of two well defined software development phases. The first one performed within an almost pure XP co-located environment, the second one involving a 20-programmers distributed team. The main goal of this experience was to show how a pure XP approach evolves while passing from a co-located to a distributed team,

doing a quantitative evaluation of the adoption of some agile practices in an open source project.

The focus was then moved in an enterprise environment with Polaris4OS research study. It was a successful experience of FLOSS adoption among chief executive officers (CEOs), managers and developers of ICT companies. The goal of this project was to fill the gap between well established proprietary enterprise processes and the upcoming FLOSS model, based on collaboration and knowledge sharing.

After this interesting beginning, the research was performed in a large scale: some of the most successful Open Source Projects on SourceForge website. The aim of the research was to investigate how and why members post in developers mailing list and the use that different members did of these mailing lists. These OS communities were studied in order to identify their main communicational practices. There were analyzed data concerning communication (mails exchange), finding interesting aspects related to collaborative learning and peer support among developers and users.

Once discovered these relevant communicational aspects, Social Network Analysis approach was utilized to analyze these popular and mature OS developers communities, in order to better understand interaction among members, individuate communicational flows and discover whether there is a sub-sequential community coordination and control. The knowledge of these relationships is useful in order to better understand how communication flows among team members and whether there is someone coordinating, controlling and facilitating informational process.

Finding these important communicational aspects is the core of this study, because they are very useful to improve, facilitate and increase the efficiency of software development. In that manner, basing on these different and articulated experimental researches, it is also possible to draw a clear and complete picture of the current situation of communicational flows in OS communities.

Chapter 1

Communication in Software Development Teams

One cannot not communicate

Paul Watzlawich

1.1 Communicating in a team

Sometimes the nature of a task determines the necessity to work in a team. It can be an intriguing challenge trying to keep together the different skills, capabilities and needs of all team members. Steiner [1], for example, in his studies examined a relevant number of teams forced to work in group to attain a specific goal. He found that, inspite of an initial reluctance, after the efficient achievement of that task, team members preferred to continue to work together only in case of similar kinds of tasks and after a positive result. In that manner a team can also decide to change its internal organization, to better cope with a particular task. To similar results arrived Triplett [2] observing athletic performances; he noticed that cyclists pedalled more quickly when they were in group, instead of whether they were alone. But working in a group is not always easy. It can be difficult finding an univocal solution or, better, a right solution. Thomas and Fink [3], in fact, found that working in a team always involves agreement searching, also in case of

wrong solutions. Some authors [4] discovered how was possible to become team leader. In fact, they noticed that, since the beginning of a task, group members used to interact more often with some better informed or proactive members on task solution. Gradually these members become team leader, playing a fundamental role, because of their central position inside the team. So it is difficult drawing a general definition of team working including all the important aspects, but an effective approach could be always concentrating the attention on the kinds of task and the specific team. But it could also be useful to remember that "*the champion team defeats champions' team*".

1.2 Different kinds of software development teams

Software development is an articulated process where is important to follow some specific activities:

- **System requirements** specify what the system has to do and its bonds
- **Development** is software implementation
- **Validation** is controlling that the system is able to do what the customer needs
- **Evolution** is modifying the system because of further needs with respect to the initial requirements

Models suited to describe software development process are based on several aspects such as main activities, temporal and logic relationships, produced manufactures and roles.
Particularly two fundamental models are very useful for a complete description of the whole development process: *Plan Driven* and *Agile*. As the name suggests, Plan Driven approach follows a thorough planning, trying to anticipate all the possible needs and further features. Instead, Agile approach is based on short interations and continuous adaptation; in fact, their saying is " *do it right the first time*". It is possible to describe these two different approach based on:

- Activity performed either by the whole team or a single member and the relationships among them
- Notation models to describe specific aspects of the system
- Documents, code and all other materials
- People's roles
- Practices and adopted method

It is not possible to talk about an univocal software development model because creating a software product is not like building a house. In a software development process, therefore, it is better to take into account different phases in order to develop, in a controlled way, quality systems. In fact, these phases are a benchmark for all the involved partners (developers, managers, customers...); phases are not fixed because every kind of project needs its specific ones. They are not necessarily done in order, so it is possible to nullify them and re-start. So, at the start up phase of a software development process, it could be useful to keep in mind these recommended phases:

- **Feasibility study** consists in problem definition, trying to find alternative solutions and their relative advantages, taking into account, for each alternative, required resources and costs. Eventually the proposal for the client is prepared based on a synthetic and preliminary evaluation; it is synthetic because there is the risk that the client changes his mind and retreats
- **Requirements elicitation** identifies requirements necessary for the application (functionalities, usability, portability...). In this phase are important *functional requirements* (they describe what the system does, with formal, informal and mixed notations), *non functional requirements* (reliability, security, interface and efficiency) and *requirements related to development process and evolution* (controlling quality and testing). Requirements can be expressed by the requirements document, use cases, user stories, formal specification methods and languages. This is a continuous process and it is better to involve always the client

- **System analysis**, also named *requirements analysis*, is useful for developers to have a complete knowledge of the domain and to avoid ambiguity in requirements expression. It is an important process that has to be done with client involvement; it utilizes specific techniques and notations, it is brought about the detailed contractual definition of the system and is the base for the design, coding and testing

- **Design** defines system architecture by sharing out the system in modules and describing relationships among them. Design and coding are related: they are always performed in a cyclic way

- **Coding** produces a system description available and executable by computer, utilizes different environment and development languages (IDE). It is important to remember that the code produces value for the customer

- **Testing** verifies the correctness of developed modules by unit tests (which verify the functioning of each system components), functional tests (which verify the functioning of the whole system simulating accesses), acceptance tests specified by the client

- **Evolution** consists in system maintenance (corrective, adaptive and perfective)

This short description of main software development process phases it is useful to understand that it is absolutely necessary a well organized model to perform all the activities and to produce a good-quality software.
For this study it is interesting to keep in mind two main models concerning software life cycle:

- **Waterfall** appeared in literature in the '50s but had a large diffusion since '70s. It is the traditional software development model where a phase starts when the previous one ends. There are specific responsibilities for each artifact, roles are predefined and communication is based on documents exchange.
  This process is quite formal and strongly predetermined, so modifying the system in an advanced development phase would be too expensive:

waterfall's saying is "*do it better since the first time*".
Waterfall is often criticized because of its strict division in phases; in that manner it is hard and expensive receiving new requirements, but it can be very useful when requirements are comprehensible and well established since the beginning

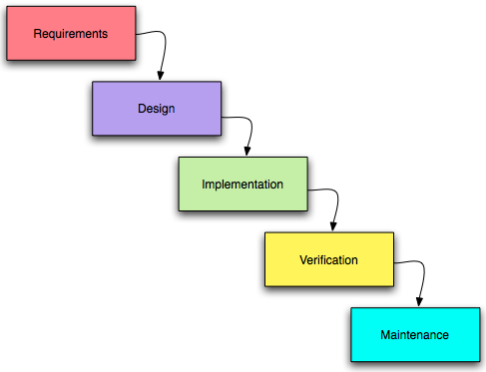

Figure 1.1: Waterfall model

- **Agile** methodologies (AM) consist in a particular approach to software development. A good way to desrcribe AM is by comparing them with traditional software development models; in fact, AM are adaptive and not predictive, oriented toward the people and not toward the process, kept simple to be able to quickly react, incremental and interactive with very short iteractions, based on testing and not on project analysis, requirements are verbal, developers play interchangeable roles, analysis and project are done in an informal way and not utilizing formal diagrams, the documentation is maintained minimal apart from the code, team members share important values.
  All these aspects suggest that the process is very simple but it must be done with strong discipline. There are different approaches within AM; they depend on the final goal, team resources and needs:

  - **SCRUM** is suited for small co-located teams
  - **Extreme Programming** (XP) is the most common
  - **Adaptive Software Development** (ASD) replaces the tradi-

tional waterfall cycle with a repeating series of speculative, collaborative, and learn cycles

- **Agile Project Management** (APM) is a specific framework broken down into five phases
- **Crystal Methods** are suited for small co-located teams working on not life-critical systems
- **Lean Software Development** is summarized by seven principles to follow according to setting and resources
- **Feature Driven Development** (FDD) consists in a mass of industry-recognized and client-valued best practices centred on functionality
- **Dynamic Systems Development Method** (DSDM) is a framework utilizing continuous user involvement in an iterative development and incremental approach

Agile Manifesto [5] is a document written by many authors personally involved in Agile methodologies improvement; they named themselves *Agile Alliance* to underline their groupthink aspect and their independent thought.
Agile Alliance fixed some relevant values to better explain their aim:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The whole team of authors was moved by some specific ideas and main themes, as the right way to approach to Agile methodologies in general, how to involve and take always into account customer's opinion, adapting software development to new requirements, releasing frequently software, involving different skilled people in the development process, preferring face-to-face communication, promoting self-organized team, meeting at regular intervals... It is interesting to underline that this

guidelines emerge from the condivision of the experience of each of the seventeen authors involved in Agile Manifesto writing because, as they asserted: "*A set of values based on trust and respect for each other and promoting organizational models based on people, collaboration, and building the types of organizational communities in which we would want to work*".

1.3 Dis-located versus co-located team: how communicating

In this short description about software development approaches and methodologies it is evident that communication plays a fundamental role at all levels: among developers, among the development team, among developers and users... As the picture above suggests. In fact, development phases are various as far as participating roles too.

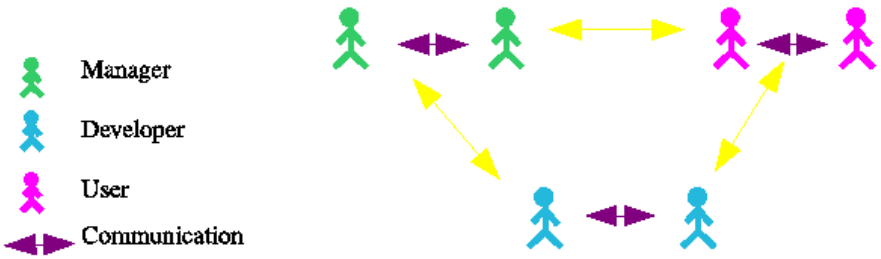

Figure 1.2: Communication in software development teams

Watzlawick [6] asserted that "*one cannot not communicate*" but it is clear that sometimes communicating could be very difficult; in fact, talking about software development teams, Beck [7] said "*What matters most in software development is communication*", underlining the difficulty to find an appropriate way to share skills and knowledge that allow an effective and efficient software development process. At the same time there are some peculiar teams (as Open Source communities) which work in a dis-located

manner and have to adapt the "traditional" ways to communicate to their displacement. So, although every approach to software development has its peculiar ways to communicate (*formal versus informal or both*), communication depends also on employed tools and, obviously, it will be very different in a co-located development with respect to a dis-located one.

Traditional software development teams are disposed only to formal communication: each communicational exchange has to be documented, each phase has its own written and established documentation, meetings have to be planned in advance.... It is clear that, because of this peculiar organization, these traditional teams could not work in a dis-located manner, in fact, it is very difficult finding appropriate tools to keep a formal communication at a distance.

Instead, AM prefer informal ways of communicating because they are more agile and friendly. So face-to-face communication in these teams is the best as far as informal meeting (like stand-up-meeting) in case it is necessary to inform and make a decision for the whole team. In AM ambit there are some teams working in a dis-located manner (for Open Source see Chap. 2). They prefer their proper communicational methods mediated by e-mail exchange, instant messaging, informal meeting, chat, forum, mailing list and wiki. Sometimes this methods replace the ones in presence, for example team members communicating in a forum do as if they are sharing the same place (the sitting together of eXtreme Programming).

This is helpful to understand that communicational strategies are very important in a software development team and sometimes, as McLuhan [8] asserted in his famous studies about human interaction, the good or bad utilization of a communicational tool can influence team performance.

Chapter 2

Open Source Communities

> We did not call our software 'free software'
>
> Richard Stallman

2.1 An important phenomenon

A book is defined classical assuming that it talks about universal matters and questions with which the man has always, since the beginning of life, had troubles. Just so, everyone thinks to have these kind of books in his library. It is the same when we talk about Open Source (OS) communities; the classical article, a milestone, in that case is "The Cathedral and the Bazaar" by Raymond [9]. He starts comparing two different development styles: *cathedral* model versus *bazaar* model, in other words commercial versus OS world.
His aim was to understand the differences between the two approaches and whether the huge success of OS projects was based on moral motivations or not. He starts from the assumption that important software needs to be built in a serious and well planned way, as handcrafts did for Gothic cathedrals. The comparison between software coding and the building of a cathedral is perfect because, in that manner, Raymond underlines not only all the accurate planning and the detailed organization standing behind groin archs,

but also the same way of working in a software house. He emphasizes the fact that, at the first glance, an OS team could be seen as "*a great babbling bazaar*" but, by studying its practices and processes, it is evident that a fundamental and central role is played by the community and the corresponding *sense of belonging*. In fact, he found that these particular kinds of communities are not moved by moral motivations (as self-help group or humanitarian organizations, for example) but they gain power from a plurality of people involved in the process. This sense of belonging and cooperation at all levels permits to take advantage of various and enormous different skills, as well as everyone's proposals and suggestions; so it is as if a huge team is working at the same project and everyone's opinion and "*ways of stressing the program*" is requested to develop an excellent software.
This short introduction is appropriate to start to understand the vast OS phenomenon but some historical specifications will be better explain it.

2.1.1 The beginning

At the beginning of Computer Age there was a quite different scenario in Computer Science respect nowadays; in fact, there were no differences among hardware and software, among users and developers, and using a computer was a sort of secret practice exercised in a mysterious way by very few people. The first revolution in this ambit was produced by the fall of hardware prices: immediately appeared the necessity of new roles and skills.
This was only the starting point of a wide diffusion of computers and the huge increasing of the related industry. And, as Weber [10] asserts, "*Suddenly the arcane subjects of operating systems and source code had moved from the technical journals to the front page of The New York Times. And open source became a kind of moderm day Rorschach test for the Internet-enabled society*".
OS is a kind of software development model with a particular licence; in fact, an interesting way to analyse software products consists in comparing their licences and distribution, in other words source code availability and cost. Based on this definition there are four most diffused kinds of software licenses:

- **Commercial** is the proprietary software; source code is not available
- **Shareware** is often free distributed in a version with some limitations and then released only after license payment; source code is not available
- **Freeware** is similar to shareware licence, but without its limitations; source code is not available
- **Open Source Software** (OSS): can be downloaded from a website and source code is available, permitting modifications and improvements. There are several OSS licenses differing for some peculiar characteristics like,for example, commercial utilization.

2.1.2 OSS between ethic and technology

As asserted in Par. 2.1.1, the revolution of OSS consists in the great opportunity of modifying the code. In fact a person could decide his own role in software development: he can simply download the software from its website and utilize it, according to his proper task; then discovering that it is lacking a particular feature and deciding to signal it or personally writing the correspondant code, it could also happen to discover a malfunctioning in the system and signal it...
Linus Torvalds's [11] words could be certainly useful to better understand this: "*This is a program for hackers by a hacker. I have enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I am looking forward to any comments you might have. I am also interested in hearing from anybody who has written any of the utilities library functions for minix. If your efforts are freely distributable (under copyright or even public domain), I would like to hear from you, so I can add them to the system*".
Once Raymond [9] has coined the term *Open Source* different abbreviations started to appear for defining this methodology: Free Software, Shared Source and FLOSS, probably the most common. It stands for Free Libre Open Source Software and it is probably preferred because unifies its double nature: both ethical and technological.

2.2 Communication in Open Source communities

Team work is defined by different factors: team size, member skills, member roles, interactions and communication.
As far as communication is concerned, some research studies [12] have found that team members involvement and performance depend also on the information provided about what it is asked to do. In some teams, as XP for example, the importance of communication is self-evident and it is well-known that it can widely change in accordance with team displacement; in fact, communication is maximized in co-located teams but, sometimes, a team has to work in a dislocated manner, as in OS teams [13] [14].
Communication in OS communities grows up around code sharing: members participate in the same software development project to improve the released product. To achieve this many practices are adopted, as information sharing, active participation in community life, bug reporting, informing other members about which new releases are essential for the success of the project. As the code is open, communication needs to be facilitated. Team members may often be dispersed worldwide, thus it is essential being able to readily communicate with one another.
In OS communities each involved person plays an important role, also considering that community members are basically volunteers, each accomplishing a different task.

2.2.1 Structure and roles

Each OS project is different and has its own peculiarities; in fact, open approach to development allows everyone to organize his own work as he prefers.
On the other hand, certain groups of people are always part of a community, constituting the base around which the project grows up:

- **Users** use the software but do not actually participate in its development. Some of them take part in the community by posting questions to the project forum or to the mailing list
- **Advanced Users** are able to understand the code and modify it ac-

cording to their needs; they maintain a close relationship with developers using the proper communicational tools

- **Bug Fixers** are mainly users who detect and report any kind of errors or bugs in the software
- **Developers** write the code and are the main source of knowledge for other people in the community. In most communities, developers do not have the same responsibilities and authority, because some of them have been working on the project since the outset; they are called core developers, and have access to the source code repository (they can commit the code)
- **Managers** are often project founders, as early developers become managers as the project grows in size and importance; they are responsible for organizational aspects, like release management and developers' work coordination.

2.2.2 Communication in OS: value and tools

In an online community, interaction among members enables them to learn through knowledge sharing, as Agile Manifesto [5] declares "*Individuals and interactions over processes and tools*" and "*Customer collaboration over contract negotiation*".
OS communities are a particular kind of online community, where developers discuss the problems they encounter in implementing a particular feature or fixing a nasty bug, and the users ask advice on how to solve any difficulties they come across when using the software, or alert the other members about errors and bugs. In the latter case, there are different tools for facilitating on line communication such as *chats*, *instant messenger*, *forum*, *wiki* and *mailing lists*.
A further tool is also often used: the developers mailing list. It is used like a (virtual) space in which developers can exchange ideas and share information about project development. In fact, knowledge sharing and free access to information are a fundamental issue for the development and the growth of these communities.

OSS can be viewed as the result of knowledge sharing among people taking part in a development community. In this sense, an OS community may appear quite similar to a development team working in a commercial software house, but the main difference lies in the fact that users also actively participate in the community: OS philosophy is based on code sharing and improvement, so each practice which can help to achieve this goal is welcome. Every OS community is organized around its particular virtual space often consisting of a discussion forum, a wiki and several other tools to facilitate communication among team members; these tools are important instruments to keep together the community and help the project to reach success.

2.3 State of the art

Apart from an enormous number of researches about technological aspects in OS communities, there is also a prolific strand concerning their communication. In fact, understanding communicational process is a good method to try to improve efficiency, productivity, cohesion and mind-group of this particular kind of teams.
A short synthesis of some of these works is helpful in drawing the actual scenario of OS communities.
It is known that the community is the fundamental unit around which the team grows up; motivation is one of these growing factors as Ye and all underlined [15] studying the importance of motivation for participating in OS communities and for becoming part of a successful OSS projects Basing on *peripheral participation theory* of Lave and Wenger [16] they found that in these communities learning process exercises a prominent role, thanks to the co-evolution between software development and peripheral participation to communication. In fact, for community members it is necessary communicating each day, improving their level of knowledge on the development process and the relationships among themselves; in that manner "*users turned developers, forming a community of practice*", and role transformation has just started. Peripheral members spend their social and psychological support to motivate all the team, and they can be considered as the audience at theatre: with clapping they support actors.

The same authors found [17] that peripheral members can improve their role becoming "*sources of the evolution of the system*", thanks to appropriate contributions, and central members "*have a larger radius of influence*". This can be easy explained with two philosophical concepts: *ontogenesis* and *phylogeny*. In fact, naive members started to participate at OSS community to solve their problems related to software utilization and, once understood how the system works, "*knowing in action*" [18] they reach a more central position. There is a co-evolution process: the naive members became skilled and central members of a community, which is well organized thanks to the contribution of its members (that at the starting point were naive members...).

Ragab et al. occupied also of members participation in a study on members proactivity among the community [19]. They analyzed communities as networks, finding that each actor is part of a decision process to "*join-leave the community network by creating-destroying its logical links with its neighbor's members based on its user's preferences*". Each member's contribution plays a bilateral role, improving both the development process and communicational efficacy, and sub-sequentially improving satisfaction of the whole team. In fact, the network performed by members allows to quickly pass information only once among the community, avoiding redundancies, decreasing the traffic among nodes and improving members' satisfaction. Other studies [20] analyze communication flow in specific OSS communities: every message posted in a discussion place (specific web page, forum, wiki..) is considered an artifact; in fact, these messages (or e-mails) are helpful to coordinate project-specific development activities, negotiate and revise relations, emerging as "*boundary objects*" [21] that allow to express software requirements and design. Scacchi [22] defines this process "*software informalism*" because these specific virtual tools let roles and their dynamics (authority, expertise, collaboration, leadership, control, conflict) to emerge, and facilitate communication among all the team, in spite of its dis-location all over the world.

The same studies found also that OSS can be interpreted also as a community "*which opens the door to access*" to a vast set of people whose contributions and discussion modify its own infrastructure, allowing a better technical and

social development. In fact, members collaborate utilizing specific tools, finding bugs, requesting software features, writing code, participating to mailing list, forums and newsletter, anticipating and solving questions and troubles. All these issues explain the good organization of these communities that are tidily coordinated by the leadership (in these communities the leader is always a developer, a module maintainer or a release manager): the only role formalized here.

In general, leadership emerges in two ways: in the field, giving technical and social support to others, implementing new ideas, managing project issues, or accepting the nomination done by others: "*to be certain, though, roles do not imply authority, but instead responsibility. Authority in the Netbeans community is based more so on reputation and respect. This authority controls manpower (what tasks get done), community infrastructure (how members interact), information availability and transparency, and representation in (and transparency of) decision-making*".

Communicational and coordination aspects, as leadership [23] and cohesion [24], are widely studied in OSS communities because they are extremely important to build a good and collaborative team with a strong sense of belonging. In general sharing the code allows to share also resources, to learn new techniques, develop new skills and solve problems improving team cohesion and its later efficiency, thanks to "*a sense of common knowledge*". This cohesion and widespread communication allow to cope with problems in a specific (virtual) place where all the members can and are encouraged to participate, increasing "*individual responsibility and self-worth regarding project activity*".

Chapter 3

MAD Project

Since now the discussion was in-centered on the importance of communication in software development teams. In fact, as previously asserted, knowing communicational aspects is helpful to get better software development process.
The word *model* comes from the Latin *modulus*, which indicates the model of a building made by architects to demonstrate to the commitment the future work: something small standing for something bigger.
A model can also be built in a metaphorical manner, as this thesis concerning OSS communities would like to be. In fact, before passing to a research study on large scale (see Chap. 5 and Chap. 7) it was considered better to accurately follow two software development projects: the first academic (using eXtreme Programming) and the second industrial (adopting OSS approach). These processes were followed since the beginning to have a complete paramount of all kinds of communication happening among members and to investigate what level of agility can be brought in a distributed context.
In this chapter is discussed the academic case named MAD(Agile Distributed Methodologies) and in the following (see Chap. 4) the industrial one.

3.1 Action research

It was previously asserted that this work is in-centered on communication in software development teams, so to reach this goal there were involved

different skilled people, playing also unusual roles, as a psychologist.
In this research also the psychologist plays a fundamental role in fact, since the starting phase of the project, she shared the space with other participants performing an *action research*.
This interesting technique was formalized by Kurt Lewin [25] in 40's, after the Second World War. In fact, USA government asked to him to persuade American families to consume also less precious pieces of meat, as offal, to try to solve financial recession.
Lewin performed a specific experiment with the aim to understand which could be the best strategy to change food-behaviour. Then he discovered that target direct observation and its involvement, in this case housewives, give immediately and lasting results (after the experiment 32% of housewives started to cook offal).
Lewin defined his approach as "*a comparative research on the conditions and effects of various forms of social action and research leading to social action*" and "*a spiral of steps, each of which is composed of a circle of planning, action, and fact-finding about the result of the action*".
Once formalized, this approach was adopted in every situation in which it was important the understanding of human behaviour to try to solve specific problems, adopt new solutions and improve efficiency, as the psychologist involved in these researches (see Chap. 3 and Chap. 4) did.

3.2 Using extreme programming in a distributed team

Agile Methodologies (AM) have been proposed as a solution to problems resulting from the turbulent business and technology environment faced by organizations engaged in software development and have been largely and successful applied in order to improve quality and to respond to fast requirement changes. AM embrace communication as a main value, as stated by the Agile Manifesto [5] "*Individuals and Interactions over Processes and Tools*". Extreme Programming (XP), for example, supports communication embracing specific practices, such as co-located team within open spaces, face-to-

face interaction among team members, direct interaction with the customer, pair programming and informative workspace. This leads to assert that an agile approach is incompatible with distributed environments where face-to-face communication is inevitably compromised.
On the other hand, nowadays distributed development models give rise to great interest. For example, in order to reduce costs and improve time to market, many companies have turned to off-shoring and outsourcing, which imply a strong distributed organization. Another successful and always more largely applied distributed model is OS. The great success of many OS projects developed by free community of developers (see Linux, Apache, KDE, Gnome, etc.) has attracted the attention of many industrial firms, which have adopted the OS process as a business model (see IBM for Linux, the Eclipse Consortium, etc.).
Despite the incompatibility, some companies have started trying to merge AM and distributed development, in order to preserve as more as possible the advantages deriving from the two approaches [26] [27] [28] [29].

3.3 Project description

The main goal of this study is to investigate how the Agile Methodologies (AM), with a special focus on XP, can be applied in a distributed environment and understand what level of XP practices adoption can be applied in such a context. In order to reach this goal it was established an academic case study. More precisely, in this work it is described an experience, performed at the University of Cagliari, in using XP within a distributed context. It will seek to demonstrate trough empirical evidence that XP values can be supported by multi-site team and it will present how to redesign some XP practices in order to fit the needs of a software distributed team.
The project consists of two well defined phases, the first one consisting of the development of a kernel set of functionalities and the second one of a set of add-on functionalities to be plugged on the kernel. The development of the application kernel has been performed by the core team, within an almost pure XP co-located environment. Rather, the second phase involves the 20 programmers distributed team. The main goal of this experience is to

understand how the pure XP approach evolves while passing from the first to the second phase. It will seek to investigate how the values and practices of XP must be modified, removed or tool-supported as the first co-located team becomes dis-located.

3.3.1 Training

The first phase was important to homogenize knowledge and skills, so team members whom need (almost students) spent three months learning the fundamental information about code writing, programming language, XP approach in order to be able to participate at development process.

3.3.2 People and roles

The case study consists of the development of an information, content and service management portal application for the University of Cagliari. The project involves 34 differently skilled people from different backgrounds. More precisely, the team is made up of 14 core members working as a co-located group, and 20 undergraduate dislocated students working independently from the core team. Among the core members there are two PhDs, one of them playing the role of proxy customer for the teacher and administrative sides, six PhD students and six undergraduate students, one of them playing the role of proxy customer for the student side. Also, the core team is supported by the work of a psychologist participating observant, who plays a role not contemplated by the XP methodology. She was called by others "The Spy". The Spy helps team to understand the dynamics of human interaction, in order to improve the quality of communication and the sense of team partnership. Each core team member plays a specific XP role (See Table 3.1).

3.3.3 Co-located development and XP practices

This first phase went on two months, it is made up of seven iterations and it is going to explain in more details how the most important XP practices were adopted during kernel development.
The XP methodology is based on a set of five values and a set of practices

Background	XP Role
Ph.D.	**Proxy Customer (Teacher and Administrative Sides), Coach**
Ph.D. Student	**Tracker, Tester, Developers (4)**
Undergraduate Student	**Proxy Customer (Student Sides), Developers (5)**
Psychologist	**The Spy**

Table 3.1: Core Team

to be applied in order to make those values explicit [7], [30]. This section describes the kernel development, explaining which XP practices have been adopted by the core team in order to create a community that embraces XP values.

- **Communication** among team members must be maximized. Among core members there is a strong emphasis on direct communication creating a sense of team and effective cooperation. The core members apply some XP practices, such as Sit Together, Informative Workspace, Pair Programming, Stories, Weekly Cycle, in order to improve communication

- **Simplicity** consists to do the simplest thing that could possibly work. In the kernel development many XP practices such as Weekly Cycle, Stories, Testing and Emerging Design from Coding and Refactoring are applied in order to help development team to do the simplest thing

- **Feedback** at different time-scale. Weekly Cycle, Pair Programming, Stand Up Meeting, Real Customer Involvement, Testing allow programmers to obtain continuous feedback on their work

- **Courage** consists in not to have fear. All methodologies and processes are tools to handle and reduce the teams' fears. In order to reach this goal the core team is supported by the following practices: Pair Programming, Testing, Simple Design and Refactoring

- **Respect** consists in not to have King and Queen. The core team applies Sit Together and Shared Code that help team-members to be respectful to their colleagues.

During this phase it was improved and tuned the adopted methodology.

- **Stories**: all system functionalities were written by two proxy customers and then estimated by developers to quantify the development effort. Until third Iteration each story was considered done when its tasks were completed, since fourth Iteration it was done only if its tasks were done and its acceptance tests were passed. This is an example of the methodology evolution

- **Informative Workspace**: in order to track the project evolution, it was decided to use both an automated tool [31] and a blackboard (see Fig. 3.1). The developers usually hang the stories on the blackboard. It is made up of three different parts: "to do", "in progress" and "done" where are collected the user stories depending on the status. This tool allowed the developers to know at any time the project evolution in detail.

Figure 3.1: Stories on blackboard

- **Pair Programming**: during the first phase of the project, the software was developed in pair programming. The Extreme Programming [30] suggests that pairs should rotate frequently. It was noted that the pairs did not rotate during whole development task. At the beginning this behavior was (partially) tolerated, but since the second half of the project, the team leader imposed the pair members exchange every development session. After a difficult, initial adjustment period, it has been observed an improvement in the communication and knowledge sharing. A proof of this is represented by a lower estimation error in

terms of effort to implement the stories. In fact, as shown in Fig. 3.2, after the fourth iteration the estimation error decreased

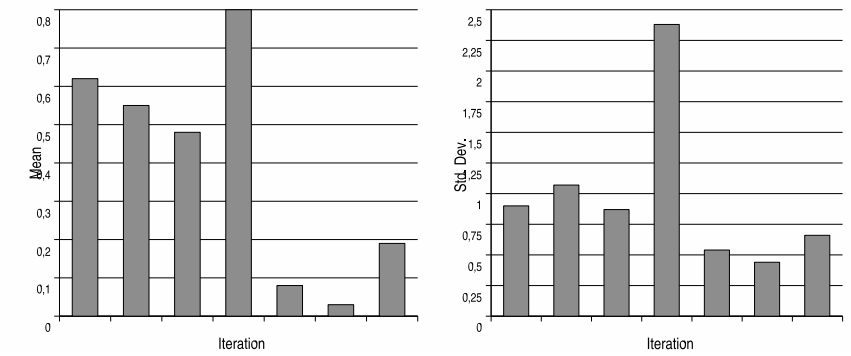

Figure 3.2: Estimation Error

- **Weekly Cycle**: the team work was planned in a weekly cycle. At the beginning of every week the core team had a meeting. During these meetings, the team leader with proxy customers picked up the stories to implement for the week. Then developers broke the stories into task and each one was able to choose a story to develop. As said above, the system kernel was developed in pair. So, a developer who had not assigned neither a story or a pair was able to choose a pair

- **Sit Together**: the kernel was developed in an open space big enough for the whole core team. The layout of the room (see Fig. 3.3) allowed the communication among different pairs

- **Incremental Design**: this practice is quite difficult to apply and requires developers to have a great amount of experience. Also it is simple to misunderstand the meaning of this practice falling into a no-design behavior. In this project, it is hardly trying to make design naturally rising from coding, always doing the simplest thing that could possibly work.

3.3.4 Distributed development and XP practices

The main objective of the second phase of the project is gradually move toward a distributed environment, while keeping XP values adapting XP

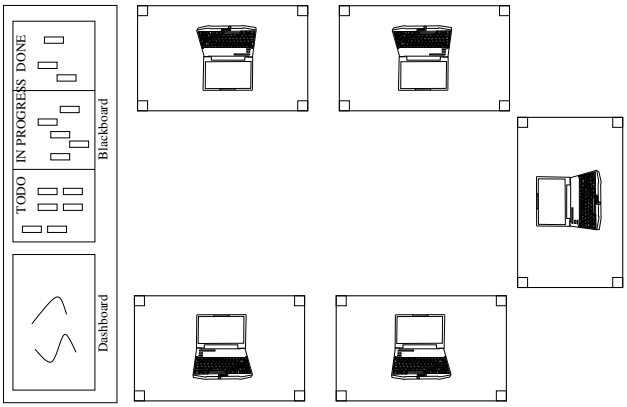

Figure 3.3: Open space setting

practices and introducing supporting tools. The following section explains how the adopted XP practices are affected by the transition from a co-located to a distributed environment.

- **Pair Programming**: while this practice represents a powerful tool for supporting communication and knowledge sharing among team members, it strongly needs the co-location of the team. Since, as there are few examples of techniques to support distributed pair programming [32], [33], it was initially decided to renounce in applying this practice. Then it was tried to use some tools such as Sobalipse [34] and Skype [35], that guaranteed a good communication level between the two developers of a distributed pair. Sobalipse allows developers sitting at different machines to edit and review the same file, and to communicate using a chat or better using the VOIP technology. One of the most important features of these tools is that they are completely integrated in the development environment [36]

- **Real Customer Involvement**: it is possible to apply this practice also in the second phase of the project because there are two developers playing the role of proxy customer

- **Sit Together**: in a sit together team all developers work in an open space in order to maximize communication among team-mates. It absolutely needs the co-location of the team so it is not applicable to a

distributes team. Anyway, there were used some tools such as a wiki, an instant messenger and a mailing list in order to favor a "sense of team" among distributed developers

- **Informative Workspace**: this practice allows the team to improve the communication. In the second phase of the project this practice is represented only by XPSwiki. The Fig. 3.4 shows the XPSwiki user interface.

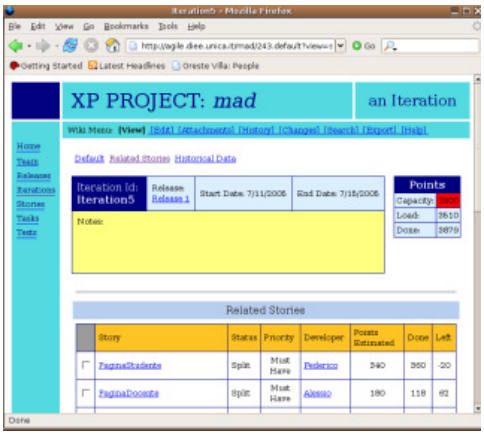

Figure 3.4: XPSwiki

- **Shared Code**: each team member is enabled to access to a SVN code repository in order to improve and change any part of the system at any time

- **Continuous Integration**: in the second phase the continuous integration is automated and performed after no more than a couple of hours

- **User Stories**: the functionalities of the system are described using stories that are published in the XPSwiki

- **Weekly Cycle**: by using the XPSwiki tool, each developer can understand which user stories are done, in progress or to do in the weekly cycle.

3.4 Co-located versus distributed communication

In some cases, as the research just exposed, the task requires a different dis-location of the same team and, to understand team dynamics and emerging roles, it is important to evaluate how changing communicational strategies in these two different modalities.
In this study it was found that, since the starting phase, more skilled members (PhD e PhD students) had a prominent role among the community. Especially one of them using to organize stand-up meeting, to write task and *to do* lists on the blackboard, and keeping an "organizational" behaviour, was recognized by others as "key member". In fact other members used to ask him information and suggestion; his role was also fundamental when it was decided to force turning of pair programming.
Day by day, thanks also to the common goal, the team was more harmonious and communication was maximed.
When the team moved in a dis-located context it was curious to note that relationships and roles were not changed and that team members tryed to adapt the provided communicational tools to maintain the same communication level in the group.

3.5 Adopting Agile Methodologies in a distributed environment: a case study simulation

Once obtained these results it was decided to understand how XP practices would have to be modified in order to adapt them in a distributed environment, trying to demonstrate trough empirical evidence how introducing some supporting tools allowed XP practices to be adopted in a distributed environment.
In fact, Agile Methodologies and distributed development are two popular trends of software engineering. The former allows to respond to requirement changes while improving the overall quality of application released. The benefits of the latter are both the reduction of costs and of development time. One of the most important examples of distributed development is open source. It has been noted that Agile processes are more effective for

co-located teams. A debate is growing up about the adoption and the effectiveness of agile methodologies in a distributed environment. So it was performed a quantitative evaluation of the adoption of some agile practices in an open source project, carrying out an academic case study with a distributed team adopting some XP practices. To evaluate the differences and the analogies among the two methodologies it was adopted a simulation approach to understand how the agile development process is affected by the distribution of the team members and how development activities are enhanced, influenced or sometimes misdirected when developers use the Internet as main coordination and cooperation tool.
The simulated project has been realized using a simulator of an XP process. This simulator has been calibrated and validated using data gathered during the first phase of the real project.

3.5.1 Method

This research follows five main steps:

1. Collection of metrics during the co-located phase
2. Calibration and validation of the XP simulator using data gathered in step one
3. Simulation of the project as if it had been performed following the XP co-located methodology
4. Collection of metrics during the dis-located phase
5. Evaluation of the differences between the real and simulated project in terms of code productivity, rate of released functionalities (User Stories) and other metrics that will be explained in more details in section 3.5.2.

3.5.2 Metrics

In order to compare the real and the simulated project there were collected different metrics, used either for the calibration and validation purpose

or the quantitative comparison between the co-located and distributed approach. In particular, context metrics are used to characterize the distributed and co-located teams and effectiveness metrics to quantify differences and analogies between the two methodologies. Among context metrics the attention was focused on *Team Size* (Number of Team Members), *Team Education Level* (Number of PhDs, Undergraduates) and *Team Skill* (Domain Expertize and Language Expertize). The effectiveness metrics include both process and product metrics. Process Metrics have been extracted from XPSwiki, a web based XP project management tool used for managing requirements, for planning and tracking activities. The tool can also be used to extract metrics such as number of stories, number of tasks per story, estimated and actual effort for each story and task, team velocity, release and iteration length and so on.

The product metrics gathered include project, class and method metrics. The project metrics are: Number of Classes, Number of Methods and Total Lines of Code. The class metrics include: Number of methods, Lines of code. Finally method metrics are the Lines of Code of each method. These metrics have been extracted from Subversion [37], the version control system that has been adopted in this case study. Subversion easily allows to examine the evolution of the source code, capturing snapshots of the project source code at any time instant. In particular it was analyzed the source code snapshots at the end of each iteration, using a system that is able to parse the source code and extract the desired metrics.

3.5.3 Calibration and validation

Agile group of University of Cagliari has developed an XP simulator that allows to forecast the evolution of an XP project implementing some of the most significant XP practices (Pair Programming, TDD, Planning Game...), and it is able to vary the adoption level of some of them. This simulator has been used to study how the MAD project would be evolved if it had been performed following the XP traditional methodology. The input parameters and the output variables that can be obtained from the simulation model are shown in table 3.2.

Input parameters	Output variables
Number of initial USs	**Number of final USs**
Number of developers	**Defect density**
Mean and standard deviation of initial USs estimation	**Number of Classes, methods, DSIs**
Initial Team velocity (pts/iteration)	
Number of Iterations per Release	
Typical iteration duration	**Each modelled entity**

Table 3.2: Input parameters and output variables of the model

Data used to calibrate and validate the simulator coming from the first release of the project. During this release the system kernel was developed. The calibration of the model parameters has been performed using data from the fifth iteration, such as the number of developers, the number of user story and so on (see table 3.3).

Input parameters	Values
Number of initial USs	**30**
Number of developers	**14**
Mean and standard deviation of initial USs estimation (pts)	**265 (220)**
Initial Team velocity (pts/iteration)	**810**
Typical iteration duration (days)	**5**

Table 3.3: Input parameters to calibrate the model

With these input parameters a number of simulation runs have been performed. In particular, it was iteratively calibrated the model parameters in order to better fit the real data of the project. In table 3.4 [1] the simulation output are compared with the ones taken from this case study.

Then, a model validation has been done using data gathered from the sixth iteration of the real project. In detail, it was changed only the initial number of stories developed (35), while it was maintained unchanged the model and project parameters obtained during the calibration of the model

[1] Comparison between simulation results averaged on 100 runs and case study. Standard deviations are reported in parenthesis. A story point corresponds to 1 minute of work

Output variable	Simulation	Real Project
Total days of Development	36.9(8.1)	36
Number of User Stories	38.2 (4.3)	39
Estimated Effort [Story points]	13697.6 (5191.1)	13252
Actual Effort [Story points]	17902.4 (4206.3)	18443
Developed Classes	77.2 (19.2)	80
Developed Methods	407.4 (100.7)	400
LOCs	3259.7 (805.2)	3260

Table 3.4: Calibration of the model parameters on the fifth iteration

(see table 3.5 [2]). It was decided to choose data from these two iterations because it was noted empirically that they were more significant than the previous ones because the methodology was not completely tuned.

Output variable	Simulation	Real Project
Total days of Development	42.5 (9.5)	41
Number of User Stories	45.3 (5.2)	44
Estimated Effort [Story points]	15680.7 (5182.5)	15442
Actual Effort [Story points]	21369.8 (5337.2)	21570
Developed Classes	92.1 (25.4)	91
Developed Methods	488.0 (134.3)	451
LOCs	3904.0 (1075.2)	3951

Table 3.5: Validation of the model parameters on the sixth iteration

It is interesting to note that the number of user stories developed at the end of the fifth/sixth iteration (see table 3.4/3.5) is greater than the number of initial user stories set as input parameter. In fact, the initial user story could be split or new user story could be added during the development.

3.5.4 Simulating the XP co-located process

After calibration and validation of the simulation model it was simulated the co-located project. There were performed 100 runs using the user stories

[2]Comparison between simulation results averaged on 100 runs and the case study. Standard deviations are reported in parenthesis

of the whole project. In table 3.6 [3] mean and standard deviation of the simulator output variables are reported.

[3]It was averaged on 100 runs on the case study. Standard deviations are reported in parenthesis

Output variable	Simulation
Total days of Development	163.9 (20.9)
Number of User Stories	196.0 (11.1)
Estimated Effort [Story points]	69291.5 (19009.5)
Actual Effort [Story points]	93109.1 (11198.2)
Developed Classes	471.7 (76.5)
Developed Methods	2492.9 (405.1)
LOCs	19940.8 (3239.3)

Table 3.6: Simulation results of the whole project

It was found that there is a big difference among the results obtained during the real development and the simulated approach, so the same methodology was applied to Apache HTTP Server to evaluate the effects of TDD (Test Driven Development) [38] but this is another research study with a goal differing too much to the aim of this study.

Chapter 4

Polaris4OS project

MAD project (see Chap. 3) was performed with particular attention to communicational practices, in order to understand how they can change and evolve according to web technologies.
A part from university staff (PhD students and PhD), people involved were almost students at their first software development experience. So it was decided to perform another project of this kind involving a different target: professional developers. This project involves University of Cagliari and Polaris (a technological and scientific park) with the aim to promote OSS adoption in their area.
In fact, FLOSS adoption in Small Medium Enterprises SMEs introduces some non-trivial drawbacks. For business men and chief technology officer (CTO), FLOSS is critical since there are not case studies and deep analysis about it, making them lost in a huge collection of untrusted reports.
On the other side, it is perceived by project managers as a critical resource, for which they have nobody to blame for malfunctioning or troubles.
Finally, developers consider FLOSS as a synonymous of "Save as..." command in their browser: FLOSS is the cheapest way to access example code and get ready-to-run libraries and applications. Both business and technical people miss most potential opportunities offered by FLOSS, like knowledge sharing, collaborative software improvement, large feedback basis, community support.
FLOSS philosophy, spirit, methodology, tools and community can be consid-

ered as an opportunity which requires skills and expertise outside the official learning path represented by universities, schools and certification centers. A new approach at learning program is needed to take advantages of the FLOSS ecosystem, as consumer and producer actor.
This work mainly focuses on an "open source curriculum" defined and applied in the Polaris4OS project. It enables developers to effectively work in the FLOSS world and encourages chief executive officers (CEOs) to take advantage of these opportunities: it implies new learning methodologies, new skills to supersede closed and proprietary approaches, new business models to optimize company resources.
The psychologist participates also in this research not only playing the role of action researcher, but also doing *tutor process mediator*, monitoring continuously participants' activities and their communicational exchanges.

4.1 Polaris4OS

The main goal of the Polaris4OS project is to fill the gap between well established proprietary enterprise processes and the upcoming FLOSS model, based on collaboration and knowledge sharing.
Eleven locally-based small and medium IT focused enterprises (SMEs) were involved in design and development of extension modules for existing open source frameworks. They had been selected according to the mission of each company and their market. Polaris4OS consists of a preliminary set-up phase and three executive phases: the training based on a blended learning model, the collaborative design and the open source distributed development.

4.1.1 Choosing a suitable Open Source project

During the preliminary phase, companies were invited to evaluate existing state-of-the-art open source projects, and were asked to select projects to contribute to.
They identified an Enterprise Resource Planning (ERP) and a Content Management System (CMS), which were considered as strategic solutions for their core business. Then the project mentors met the companies to design and plan the next phases.

Their core business had been totally based on proprietary software, with both custom solutions and off-the-shelf applications. Moreover, they were competitors in the same geographic area and they never cooperated. They had different experiences with a mixed background of Java and .NET environments, and each had its own software engineering methodology, usually based on practice and not on a formal approach.
Preliminary meetings let to set-up a trusted environment in which companies agreed to share their experience, knowledge, resource, code and developers. The mentors identified the skills and the main competence areas needed to participate to selected projects. The key technologies were deemed to be object oriented programming, integrated development environments, collaborative utilities, versioning systems; the most important methodologies were considered to be design patterns, test driven development, continuous integration; the key processes were deemed to be collaborative work and frequent iterations.
The project effort was established in two man-days a week per company, for a timespan of nine months.

4.1.2 People and roles

There are different roles involved in this project, as it is possible to see in Table 4.1.

Role	Description	N
Mentor	They identify the skills and the main competence areas needed to participate to selected projects and organize meetings with the companies to design and plan all the phases	5
CEO	They are competitors in the same geographic area and they have never cooperated before	11
Developer	They are employees of the involved companies	18
Customer	They play the role of real customer in order to achieve better results to final user	2
Teacher	They learn fundamental concepts related to software development	12
Coaches	They are skilled people trying to help developers and facilitate development phase	2
Tutor	She facilitates communicational process among participants both in presence and in blended modality	1

Table 4.1: Different Roles

All these different skilled and motivated people compose the development team and worked together.

4.1.3 Training

Since the initial developers skills were very different, it was decided to adopt a flexible learning method to support a training process, mixing classroom lesson and laboratory activities, on-line training and support, collaborative learning among learners, coaching and tutoring [39].
Learning model answering to these necessities is the blended learning one (BLM) [40] [41]. It was supported by Moodle [42] learning management system (LMS) and by the collaborative knowledge base, realized with the learning objects published by teachers.
This research model consists of three main learning moments:

- **Collaborative learning** is based on specific and practical activities during frontal lesson
- **Self-learning** gives to developers the possibility to use the material loaded in the LMS by teachers and to interact with coaches and tutors

- **Consolidation**: developers came back in classroom to demonstrate the achievement of the administered concepts

The BLM was realized in:

- 4 hours of classroom **frontal lessons**
- 8 hours for individual study, collaboration, exercises based on the **learning object** prepared by teachers and having the support of tutors and coaches
- 4 hours for classroom knowledge and **capabilities assessment**

The LBS monitored developers activities, as the time spent by each one on it and the result of on-line tests, for example. From their analysis, it was defined a training process tailored on individual necessities, creating a sort of learning customization; it was also noticed that the developers spontaneously create a community where people helped each other and shared materials and information [43].
The community created in the first phase represents an important result for the project because it showed that the initial formers diffidence was overcame since the starting phase.

4.1.4 Designing

In such phase, there were analyzed the selected FLOSS projects and the definition of two specific application scenarios. To have the execution phases like a real project, it was decided to have real customers therefore there were selected the local administration of Pula, in Sardinia, and a private breeding company.
Meetings were taken to know customer needs and the software and hardware solution they used.
It emerged the interest of the first customer for a tourist event alert system, while the second one would like an automatic tracing system of animal growth for a pigs farm.
Tutors and coaches drove developers to evaluate the software solution and they found in Infoglue CMS [44] the most suitable FLOSS software to be

extended including the module required by the local administration and in [45] the ERP to extend and modify in way to implement the tracing system. Once selected the software, developer team worked to define the tasks and a preliminary TO-DO list.
Each item list was assigned to a developer. Design activities were mainly conducted remotely by means of chats, web forum and mails, with weekly interactions for requirement definition, tasks identification and assignment.

4.1.5 Module Development

Developers implemented the extension modules driving their activities with concepts learned in the first phases and using the results of the design process. The development model was based on continuous integration and frequent releases in order to verify the correctness of the implemented software and the referential integrity of the required changes respect to the original databases structure.
During weekly meetings, the encountered problems had been analyzed and the possible solutions discussed by all developers for the adoption of the best choice. In such meetings, coaches and tutors applied a learning-by-doing model [46] in way to address the developers lacks raised in the course of this phase.
A set of problems was related to the inexperience of the development FLOSS model, like the remote and distributed collaboration model and the possibility of reusing the source code and the databases of the selected software.
For both of modules, the first development task was the installation of the original software and set-up of the integrated development environment (IDE). There were not available automatically procedures for the installation then the developer executed in separately steps the DBMS installation, the database creation, the Compiere or Infoglue installation and the connection between database and these software.
Regarding to the IDE, the selected one was Eclipse [47]. The following tasks were tailored on the module. The alert system for Infoglue required the mapping of the new tables on the databases structure using Hibernate, the definition of the template for the web pages and the alert system. The system

function realized were user registration, session and message management.

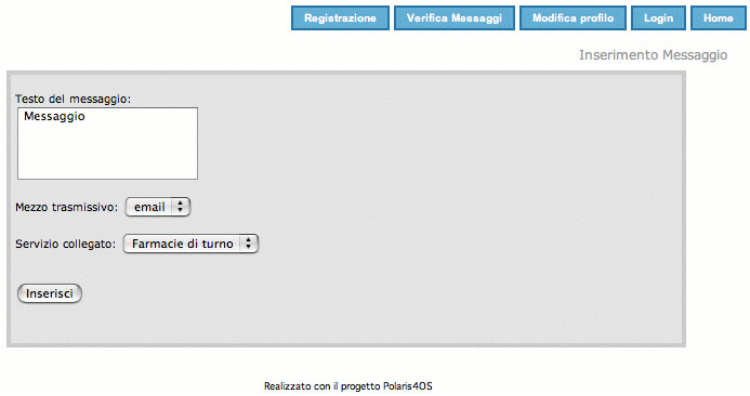

Figure 4.1: The web page where to insert messages

4.1.6 Companies participation

Companies involved in Polaris4OS come from different IT market areas and they have different size. Most of them are focused on services for Public Administration, while others work in the Internet context, with web services and application, mobile solutions, digital television and professional training. Average team size is 3-4 developers for each company.

Before starting the project, each company presented itself to other participants, in order to better identify resources provided to the project and expected results. Technical team had experience in database engines, object oriented programming language and some web framework experience. Therefore, there was not an homogeneous expertise nor similar objective for those companies. Most important, there were no previous experience with OS projects.

CTOs have been introduced to FLOSS ecosystems, in order to provide them with all elements to decide what technology to adopt and what kind of OS project to work for. They have been preliminarily introduced to FLOSS licenses, their scope, their limits and their adoption across leading OS projects. Companies have been encouraged to participate Polaris4OS because they would perceive the training phase as a valuable benefit for their team: they were able to gain new skill at no cost. However, they kept some doubts

regarding actual developers effort: it was expected to be two days a week and they considered it too short to achieve a relevant product but, on the other side, they were unable to assign more resources to the project.

4.2 Curricula

An important and none evident aspect of the Polaris4OS project was the definition of developers curricula. Teachers, tutors and coaches guided their actions in the designing and development phases in way to fit it. The curricula definition started with the training phase and it was adapted to achieve four main goals:

- All developers had to work on the project
- All developers had to use the selected technologies
- Collaborative approach to all project activities
- Build a developers shared knowledge, practices and learning community for all the following project phases

It was also defined a list of fundamental competences for the curriculum and grouped them in four categories:

- Information seminar
- Software engineering methodologies
- Use of tools and IDE (Java, Eclipse, SourceForge...)
- Features and constraints of the two selected open source projects (local administration versus tracking system)

Defining curricula was based on what can be used to design courses and curricula concerning Agile development [13].

4.2.1 Defining Curricula

The preliminary analysis identified the training goals selecting the areas of knowledge and skills; teachers, coaches and tutors needs to focus

the lessons and planning what designing and development aspect had to be treated. The output of this activity has been the definition of the curricula based on the needed skills and started from the owned skills in way to fill the competence gap.
Developers meeting evidenced the need to keep confidence with development methodologies like advanced development practices (for example design patterns), database management, data abstraction and web applications. They had not experiences on software development in distributed environment so they did not know practices like test driven development (TDD), continuous integration, unified modeling language (UML) design and related tools.
Furthermore the developers need to study a ERP and a CMS. In 4.2 are listed the identified modules and their correspondent categories.

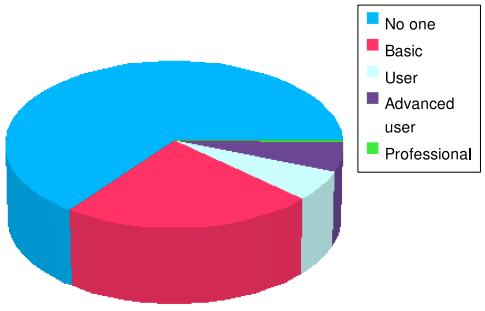

Figure 4.2: Initial developers skills

4.2.2 Results of Blended Learning Method

Developers participated actively to activities on Moodle platform and its knowledge base. It is interesting to note that they assumed a proactive pose among the community. Many people had loaded appropriate material for other formers, they share information, suggestion and doubts with on line coach and tutors. Problems was solved all together according to the collaborative learning. Developers appreciated the platform tools as much as web forums and chats.

At the end of the third executive phase, a test was submitted to devel-

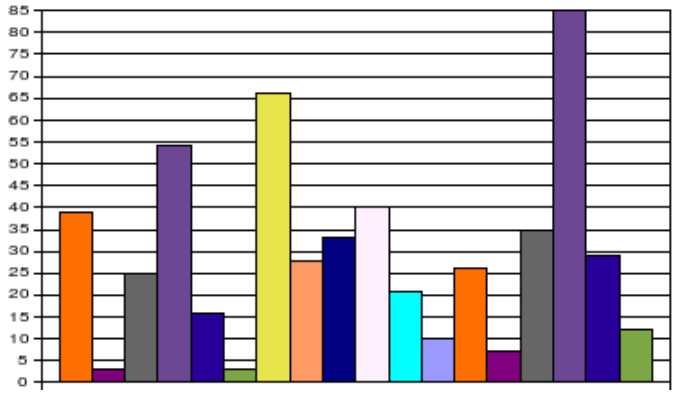

Figure 4.3: Number of visits to Moodle platform for each developer

Module Name	**Category**
What is Open Source: Open Source licenses and philosophy	Seminar
Collaborative Environment and LMS: Moodle and its tools	Seminar
Eclipse Base: basic use of Eclipse	Tools
Java Advanced: design Patterns	Tools
Eclipse Advanced: developing application using Eclipse and CVS	Tools
UML and tools: diagrams	Methodologies
TDD e Refactoring: TDD, Junit, HttpUnit, Frequent release, Short iteration, Refactoring	Methodologies
Web Application Development: Tomcat, Velocity, JSP	Tools
DB and Data abstraction: data persistence, DAO, Hibernate	Tools
Distributed architectures and programming: Client/Server programming, Java network programming, XML standard	Tools
Infoglue CMS: system architecture and technologies analysis	OSS
Compiere: system architecture and technologies analysis	OSS

Table 4.2: Curricula modules and their correspondent category

opers to collect their sensations on the whole project.

Among developers 82% of them provided a positive judgment: 62% evaluated the learning process as *appropriate* and 20% considered *appropriate at all*. Nobody found it *inappropriate*. On such results, it can be asserted the contents tough were appropriate to the target.

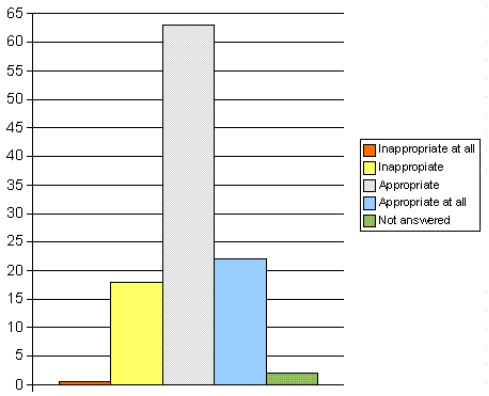

Figure 4.4: Final test results

4.3 A success experience

Some works have exposed the use FLOSS like a learning and model environment, for example [48] [49] [50]. The results of Polaris4OS [51] enforces such considerations. It has been successfully completed and it can be considered a very interesting case study to evaluate overall impact of FLOSS adoption in SMEs. Both company managers and their developers appreciated the FLOSS model and the concrete opportunities it gives. Working with real-world OS projects, they learned how to enhance their business, products and skills by means of FLOSS methodologies and practices. However, migration to FLOSS requires a focused effort to be understood, accepted and implemented.

Learners participated actively to curriculum definition negotiating topics with teachers in order to both satisfy project needs and reach their own expectations. It was a sort of negotiation about the pros and cons of each proposal and it is important to highlight that the last two curricula modules, focused on Compiere and Infoglue, were required by CTOs and developers, thanks to the continuous communication among themselves and all the people involved in this project.

Chapter 5

A research study on Open Source Communities developers'mailing lists

Nomina sunt consequentia rerum

Justinian

5.1 Some of the most successful Open Source projects

The two cases study of distributed development previously described (see Chap. 3 and Chap. 4) have make clear the process nature of communication. In fact, people roles emerge by the field, thanks to a mix of skills and information concerning a better knowledge of software development process, a good control of communicational tools and a natural ability for human interaction.
So once analyzed the two different cases study and understood the relevance of efficient communication among team members, it was decided to investigate, and consequently improve, communicational process on large scale.

5.1.1 SourceForge website

The attention was in-centered on some of the most successful Open Source projects active on SourceForge.[52]. It is a website which contains different tools to facilitate software development process. There are more than 74.000 projects utilizing it since the starting development phase, *"SourceForge.net is the world's largest Open Source software development web site, providing free hosting to tens of thousands of projects. The mission of SourceForge.net is to enrich the Open Source community by providing a centralized place for Open Source developers to control and manage Open Source software development"*.
This website is available also for users, they can download software simply entering in the specific area; instead, to utilize its tools, developers had to compile a specific form and log-in.
Once obtained id and password a developer can create its own project or start to collaborate to other projects: there are more than 750.000 developers registered on SourceForge.
There are different tools available to permit software development, project management and software advertising.

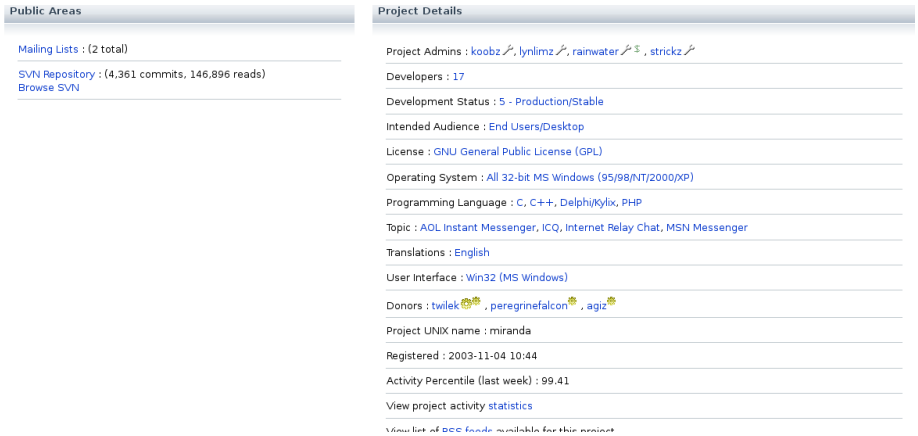

Figure 5.1: Miranda home page on SourceForge website

Once inserted the project on SourceForge team members will be, automatically, members of this huge community, in fact, every project has its own website where is possible to edit files, documentation and task to do

list, bugs list and patches to be implemented; there are also different tools like thematic forums and mailing lists to keep in contact and ask for help and information among different community members.
An important resource of this web site consists in a Concurrent Versioning System (CVS); it implements a controlling system informing every member about all the changes occured in code writing and software implementation, permitting to manage control versioning and "opening the source code" to developers interested in collaborating. CVS can manage also different branches of the same projects allowing software customization.

5.1.2 Choosing most succesful OS projects

On SourceForge there is a classification that defines which are the most active projects, based on indexes like activity, lines of code and number of download so it was decided to analyze communicational flow in most active SourceForge communities.
After a short paramount about communicational tools available for each project, the attention was focused on developers' mailing lists (DMLs). In fact these virtual place is the point of contact among developers and users, as it will shown in Section 5.1.3, and is amazing important to find communicational patterns and networks.
There were examined more than 70 projects in order to find a sufficient number of projects. In fact, looking for developers' mailing lists, it was found that most of these projects had none, as it is shown in Tab. 5.1.

Status	Projects
Corrupted archives	Crystal Space 3D, SquirrelMail, Jedit, Icewm BO2K, Mesa3D, Small Device C Compiler, Firebird, BZFlag, User-mode Linux kernel port, Etherboot, Enlightenment
Without developers ML	WebCalendar, Numerical Python, Exult, wxWidgets, MiKTeX, Tcl, Doom Legacy, AWStats, net-snmp, Firewall Builder, The Nebula Device, PhpWiki, CMU Sphinx, Courier Mail Server, gnuplot development, Developer's Image Library, Dev-C++
Without ML	Ghostscript, Python, Linux PCMCIA Card Services, Mailman, Quanta, The Freenet Project, Openads, Common C++ Libraries, Gnucleus Gabber, Boa Constructor, Scintilla, The EDGE Project
ML not active enough	Double Choco Latte, TUTOS, Slash, Screem, Leo, Owl Intranet Knowledgebase, WebMail-Java, Halflife Admin Mod, Cdex, DynAPI
Available archives	Gaim, Gimp, Licq, Miranda, MinGW, Netatalk, Gallery, Arianne, RPG Geotools

Table 5.1: The 70 most active projects on Sourceforge

This fact can be easily explained considering the kind of use that software development teams make of SourceForge [53].
Many teams started to use SourceForge at the beginning of their projects, but later, as the project size rapidly increased, they preferred to use their own website with specific tools (CMS, wikies, forums...) in place of SourceForge. At the same time, a large amount of these projects uses SourceForge website as a means for making their product better and more widely known, so, it is very likely that they use a developers' mailing list not available on SourceForge web site.
Only 9 projects out of 70 seem to actually use SourceForge developers' mailing list. These projects [1] are shown in Tab. 5.2.

[1] All of them have activity level at 99,99%

Project	Topic
Arianne	Multi player on line engine to develop games
Gaim	Instant messaging application
Gallery	Web based photo gallery
Geotools	Open source Java GIS toolkit
Gimp-Print	Package of printer drivers
Licq	Instant messaging application
Mingw	Tool for importing libraries and header files
Miranda	Instant messenger application
Netatalk	Daemon for sharing files and printers

Table 5.2: The nine studied projects

Starting from this nine successful projects it is possible evaluate communication among their members.

5.1.3 Mailing Lists Developers

In "classical" agile projects, communication is maximized through face-to-face conversation among developers and users. "Classical" open source projects, on the other hand, are performed through the Internet, and communication has to be performed using Mailing Lists (MLs), wikis and messaging tools. This condition is clearly sub-optimal under the agile perspective, but it is unavoidable. On the other hand, the communication performed through MLs can be easily tracked and studied.
Every OS project has its own developers mailing list which only programmers should use to discuss project related development. Many projects have a public developers ML which advanced users actually use to obtain help and exchange feedback among themselves and with developers.
The aim of this study is to identify and evaluate the utilization of developers' mailing list of mainstream Open Source projects, to get insight on communication patterns and practices among developers and users of the list, finding some peculiar aspects of these communities.

5.1.4 Members

DML is frequented by two different kinds of people: advance users and developers. They are basically volunteers, each playing a different role: users are also contributors; this is fundamental for the developers to implement new features and receive feedback. This involvement of end users plays a relevant role in agile methodologies and in particular in eXtreme Programming [30]; it is both a value (*feedback*) and a practice (*real customer involvement*). Developers often discuss among themselves about implementing features and fixing bugs, while users exchange opinions about their problems concerning software utilisation. In Tab. 5.3 there are underlined DMLs members' role for all the analyzed projects.

Project	Users	Developers	Participants
Arianne	90	12	112
Gaim	1195	52	1247
Gallery	431	53	484
Geotools	299	96	395
Gimp-Print	581	46	627
Licq	598	13	611
Mingw	62	43	105
Miranda	164	16	180
Netatalk	695	43	738

Table 5.3: Members and roles

5.2 Data collection and analysis

In order to investigate the developers' mailing lists chosen it was implemented a specific parser that enabled automatic data analysis. The parser was written in Java and it was created to extract key data from the repository of developers' mailing list.

In particular was made a descriptive statistical analysis on three different indicators: *e-mails*, *threads* and *links* among participants. For each e-mail, there were extracted the sender, the subject and the time, and for each

thread the ID of the starter. Different queries can be made to interrogate the repository.
Preliminary, it was checked and resolved e-mail addresses and user names: about 50% of community members (both developers and users) use different user names and e-mail addresses to post to the same mailing list (there are people who use even five different identities!).

5.2.1 E-mails

For e-mails traffic there were extracted different data: number of senders, average mean, variance, percentage of e-mails sent by developers and relative variance, percentage of e-mails sent by users and relative variance, percentage of e-mails sent by most active member, role of the most active member, number of e-mail sent by developers and relative variance, number of e-mail sent by users and relative variance, as it is shown in Tab. 5.4 and Tab. 5.5 and Tab. 5.6.

Project	Senders	E-mails	Mean	Variance	Most Active
Arianne	112	1159	11.36	1130.5	User
Gaim	1247	8954	7.18	1656.9	Dev
Gallery	484	3997	8.26	1224.2	Dev
Geotools	395	11076	28.04	12721	Dev
Gimp-Print	627	7816	12.47	16030	Dev
Licq	611	4715	7.72	1177.7	Dev
Mingw	105	2690	25.62	4338.2	Dev
Miranda	180	1045	5.81	98.85	Dev
Netatalk	738	4678	6.34	586.18	Dev

Table 5.4: E-mails

Neither users nor developers are consistently the most active in sending e-mails, as it is shown in Tab. 5.5 and Tab. 5.6.

Project	% E-mails sent by devs	Mean	Variance	% E-mails sent by most active dev
Arianne	27.87	27	2753.9	16.22
Gaim	46.8	80.58	32390	1.23
Gallery	56.34	42.49	9532.9	15.26
Geotools	65.93	76.06	37986	13.22
Gimp-Print	67.96	115.52	20872	39.14
Licq	31.22	113.23	33898	13.83
Mingw	81.26	50.84	9346.9	16.58
Miranda	30.53	19.94	499.4	6.12
Netatalk	49.36	53.7	6474	6.39

Table 5.5: E-mails sent by developers

In fact, the e-mails sent by the two subgroups change tendency for each project: in four projects users' sub-group is the most active, in another three the most active is the developers' sub-group, while in the remaining two projects the amount is evenly distributed, as it is clarified in Fig. 5.2.

Project	% E-mails sent by users	Mean	Variance	% E-mails sent by most active user
Arianne	72.13	9.29	905.56	23.3
Gaim	53.2	3.99	100.7	12.97
Gallery	43.66	4.05	60.13	2.55
Geotools	34.07	12.62	3728.3	5.69
Gimp-Print	32.04	4.31	198.31	2.8
Licq	68.78	5.42	274.31	5.39
Mingw	18.74	8.13	201.59	2.83
Miranda	69.47	4.43	41.08	3.73
Netatalk	50.64	3.41	83.14	2.69

Table 5.6: E-mails sent by users

This matter will be analyzed in further investigation (see Chapter 7).

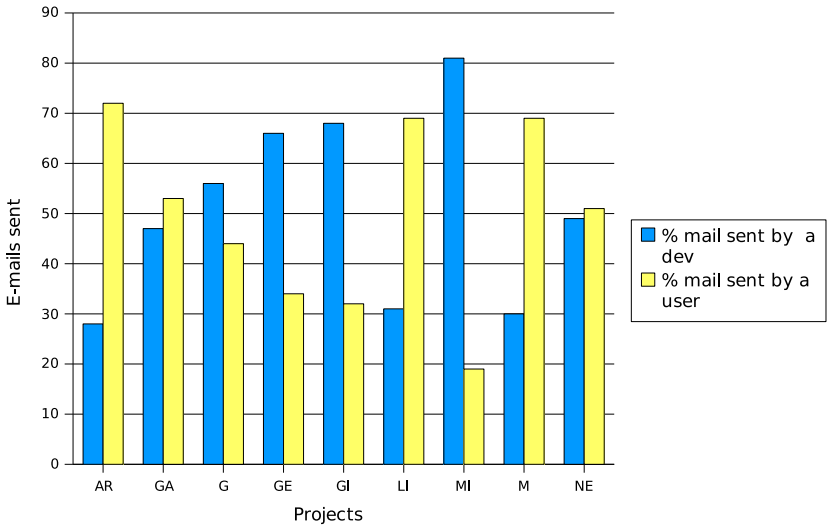

Figure 5.2: Mails sent in each project by developers and users

There are some developers who send from 5% to 20% of the total number of e-mails: just a small number of developers is the main suppliers of e-mails in their DMLs.

These active developers play a *key role* in the communication process, in fact, they are either project managers, support managers or developers. Their *key role* [54] is to know everything about the project and to share information and provide explanations to users.

5.2.2 Threads

For each thread there were extracted different data: number of senders, average mean, variance, percentage of threads started by developers and relative variance, percentage of threads started by users and relative variance, percentage of threads started by most active member, role of the most active member, number of threads started by developers and relative variance, number of threads started by users and relative variance, as it is shown in Tab. 5.7, Tab. 5.8 and Tab. 5.9.

Project	Threads	Mean	Variance	Threads started by most active member	Most Active
Arianne	452	4.43	334.35	175	User
Gaim	2605	2.08	131.95	369	Dev
Gallery	1325	2.73	85.06	131	Dev
Geotools	4313	10.92	2497.6	744	Dev
Gimp-Print	1871	2.98	601.38	600	Dev
Licq	1559	2.55	46.63	87	User
Mingw	647	6.16	255.58	94	Dev
Miranda	229	1.27	4.18	14	Dev
Netatalk	1672	2.27	43.53	98	User

Table 5.7: Threads

Developers, on average, start about 35% of all the threads, as it is shown in Tab. 5.8. In these threads, developers use to announce new releases, report a bug and discuss new features. It is important to note that, in each project a single developer, on average, starts a number of threads ranging from 5% to 30% of all threads.

Project	% Threads started by devs	Mean	Variance	% Threads started by most active dev
Arianne	21.24	8	219.82	11.72
Gaim	34.18	17.04	2849.8	14.23
Gallery	47.39	11.81	640.69	9.92
Geotools	58.94	26.5	6447.4	17.25
Gimp-Print	52.94	21.52	7906.3	32.08
Licq	16.02	19.23	498.86	4.87
Mingw	82.84	12.47	551.35	14.53
Miranda	21.83	3.13	15.18	6.11
Netatalk	27.27	10.6	340.67	5.38

Table 5.8: Threads started by developers

These developers are also the *key members* (see Paragraph 5.2.1), confirming again the fundamental role of those members.
On the contrary users, usually, start about 65% of all the threads, as it is shown in Tab. 5.9.

Project	% Threads started by users	Mean	Variance	% Threads started by most active user
Arianne	78.76	3.96	350.31	38.71
Gaim	65.82	1.43	5.8	1.19
Gallery	52.61	1.61	6.65	1.66
Geotools	41.06	5.92	1143.5	12.59
Gimp-Print	47.06	1.51	6.24	1.71
Licq	83.98	2.19	31.43	5.58
Mingw	17.16	1.79	8.69	2.32
Miranda	78.17	1.09	2.82	4.8
Netatalk	72.73	1.75	21.03	5.86

Table 5.9: Threads started by users

In these threads users ask for information, answer other users' questions when they solved the same or a similar problem and, sometimes, report a bug.

5.2.3 Links

A link is a connection between two members belonging to the same developers' mailing lists. Two members share a link if they have participated in the same thread.
For each link there were extracted different data: number of links, average mean, variance, link among developers and its percentage, link among users and its percentage, link among developers and users and its percentage, mean average of links for each developer, each user and among developers and users, as it is shown in Tab. 5.10 and Tab. 5.11.

Project	Links	Links whitin developers	Links whitin users	Links among members
Arianne	370	24	229	117
Gaim	5550	280	2414	2856
Gallery	1667	244	610	813
Geotools	3162	529	1790	843
Gimp-Print	1490	181	387	922
Licq	1896	32	1188	676
Mingw	509	201	66	242
Miranda	1091	34	681	376
Netatalk	1632	226	520	886

Table 5.10: Links

Analyzing links between community members is helpful to understand communication [55] among team members and their consequential effort and involvement in the project.
Links between users are about 60% in four projects, while they range from 13% to 43% for the remaining ones. This does not mean that each user communicates, on average, with many of other users.

Project	Links of each developer	Links of each user	Links of each member
Arianne	2	2.54	1.15
Gaim	5.38	2.02	2.29
Gallery	4.6	1.41	1.68
Geotools	5.5	5.99	2.13
Gimp-Print	3.93	0.67	1.47
Licq	2.46	1.99	1.1
Mingw	4.67	1.06	2.3
Miranda	2.13	4.15	2.08
Netatalk	5.25	0.75	1.2

Table 5.11: Links of each member

In fact, observing a single thread it is evident that each user communicates, on average, only with two other users, as it is shown in Tab. 5.11; so it is possible state that a lot of users participate in communication in DMLs but only a few of them keep in direct contact.
The network formed by user communication is very poor and scattered and it was found a similar behavior in communication among all community members (users and developers together). On the other hand the links between developers are about 15% for four projects, about 5% for the other three projects, with two outsiders too: one with 39% and the other with 2%.
If a thread is singly analyzed it is possible to found that a developer communicates, on average, with three other developers. This suggests that there is a dense communication net and a great deal of information sharing among these developers, who are also the *key members*.
The network formed by their communication can be considered like an entity: the core of the communication.

5.3 Communication flow in Developers' Mailing Lists

Justinian said "*Nomina sunt consequentia rerum*" (Names are sequent to the things named), so analyzing these developers mailing lists it was initially expected to find e-mail exchange only among the project developers themselves. But surprisingly both users and developers post there and a significant percentage of e-mail traffic is to be attributed to users.
To analyze the data the community was splitted into two complementary sub-groups: developers and users. In fact users utilize this space to spell out their problems concerning software utilization and to receive explanations on the software developed (but also *development*); at the same time, developers seem to understand and support this vision of the developers mailing lists, providing suggestions and information.
Furthermore, the support developers (or more skilled members in general) are able to give to less experienced participants, can be equated to software supplier support in the case of proprietary software. It is widely recognized that one of the greatest downsides of closed software is the strong customer-supplier dependence as far as maintenance, update and support is concerned.

The learning community can easily solve this problem, because developers as well as other users can perform the same task partcipating at these projects success, in fact, it is known that the quality of support is one of the success indicators of an Open Source project.

A new learning community composed of advanced users and developers is thus formed, users or developers alike sharing information, making suggestions, asking for help, ensuring knowledge flows regardless of their *role*. In particular, a *peer support* system can be established between advanced users who help each other, while for strictly development-related topics, programmers usually communicate using other tools, like instant messaging, private e-mail, wiki...

Data analysis revealed that not all the DMLs behave in the same way about sending e-mails, starting threads or establishing relationships. Some developers are key members of the community sharing information with all other members.

These key members keep in contact with each other creating a dense communications network and users create a scattered and weak network, this kind of distribution seems to suggest a dichotomous communication pattern in which the core is composed of developers and the periphery of users. So in Chapter 7 the same data will treated in a more specific manner in order to find communicational flow and roles utilizing Social Network Analysis approach.

Chapter 6

Social network Analysis

6.1 Social Network Analysis approach

Social Network Analysis (SNA) was born in Social Science ambit, becoming a fundamental approach to characterize relationships among social entities, their development and their further implications.
SNA in fact, defines social environment in a new and different way, relationships among people are defined *structure*, becoming virtual weave crossing with all other weaves; structural variables are extremely helpful to clarify enormously different kinds of relationships among these structures.
Structural variables focus their attention on dyads, tryads, subgroups or large groups.
SNA is also helpful for analyzing networks in two particular ways:

- **Formal description**: network analysis helps to express theoretical concepts giving formal definitions, measures and descriptions; evaluating models and theories concerning relational processes and giving statistical analysis of multi-relationships systems. In that manner is it possible to share a common language and definitions to better express theoretical concepts and features
- **Models and theories evaluation and testing**: network models are also useful to prove theories concerning relational structures and processes. In fact, observing a network it is possible to make inferences

about relationships structure. As Gestalt theory asserted "*the whole is more then the sum of single parts*" and SNA is perfect to consider relationships among group members like an entity, a whole.

There is an enormous number of statistical indexes, roles, measure units and specific terms surrounding the vast world of SNA, so the most important of them related to this research work are going to be explained here.

6.1.1 Actor

One of the goals of SNA is to understand relationships between members: the actors. The term *actor* can define, for example, a person or a group of people, an employee or a person working in an organization, an enterprise department or a public services company.
Usually a research is focused on one specific kind of actors: children, engineers, housewives...

6.1.2 Link

A link is a simple connection between two actors. Actors can be linked among themselves in an illimitated range of different ways; for SNA is not important the nature of this link but its existence.
These links can be due to different motivation as evaluation of a person, like a working situation, to be members of the same association, behavioural interaction, status change, physical connection, formal or biological relationship, for example.

6.1.3 Dyad

A dyad is a particular and strong link connecting two people who share it in equal manner. They are considered a unit because of that link and this is also often the fundamental unit of a research utilizing SNA approach.
In literature one of the most studied dyad is the one composed by mother and child.

6.1.4 Tryad

The definition of tryad is equal to dyad but referred to three people sharing this important relationship. For tryads are important two properties:

- **Transitive property**: if S is a friend (in this case friendship is the link) of M and M is a friend of E then S and E are also friends
- **Tryad balancing**: if S and M think each other they are beautiful people then it is highly probably that they think the same about E

6.1.5 Subgroup

A subgroup is a set of people strongly connected among themselves; individuating and studying this particular kind of group is surprisingly important for SNA.

6.1.6 Group

Usually SNA is in-centered on a specific group and its aim is to understand links, relationships and connections among actors; often these members share some peculiar properties that allow them to be considered as an entity.

6.1.7 Social Network

A Social network is defined by the sum of actors, their links, relationships and features.

6.2 Centrality and prestige

Individuating the most prominent actor(s) in a social network is important to draw a more precise picture. An actor becomes famous if he has been chosen by the majority of other members so it is important to understand why this happens: centrality and prestige indexes are useful to do that.

6.2.1 Centrality

Prominent actors are often involved in a relevant number of links and have a visible role among the community, as centrality index measures.

6.2.2 Prestige

Talking about centrality index, it was affirmed that an actor plays a prominent role among the community because of his links, without explaining whether this centrality is due to preferences (in links creating) given or received. When an actor has the preference of a huge number of other members he plays a prestige role.

6.3 Centrality indexes

These helpful indexes are *non directional indexes*, meaning that it is known that there is a link but its direction and source are unknown or non relevant.
Centrality indexes are used in this research to better explain communication flow in DMLs as it will be shown in Chap. 7.

Chapter 7

Analysing the Social Networks constituted by Open Source communities

Ο α′νθρωπος εστι′ ζω′ον πολιτικο′ν

Aristoteles

In previous studies [56] [57] on OSS communities analyzed in Chap. 5, it was found that the virtual place where developers meet users is the DML. In fact, this place is used to share information, to gather all users' needs and proposals (feedback), and to create a link between users and developers (real customer involvement).
As the research continued, it was discovered that the above described behaviors also distinguish *learning communities*. DML is a big stage where actors-community members communicate, in fact, interaction does not necessarily take place just among peer members. Indeed, discussions among developers and users (for example help requests) are quite frequent, and allow to create a network containing numerous hubs (members who are able to establish a lot of links). This facilitates information flow through the network, and explains why OS communities can be regarded as a typical example of learning communities.

Communication among advanced users has been defined as *peer support*, while there were called *collaborative learning* and mentoring[1] and any kind of help that a developer can give to a user. All these kinds of communication among community actors has been recognized as a fundamental value in Agile teams and have been widely studied; in fact, as Beck asserted [7], "What matters most in team software development is communication".

7.1 Social Network Analysis applied at OSS communities

Social Network Analysis (SNA) approach is here adopted to describe the communities analyzed in Chap. 5 in a quantitative manner, and to better evaluate their communicational processes.
As it is known by people involved in this research, this is the first time that such a quantitative analysis utilizing such powerful mathematical tools is applied to OS communities.
This research focuses on considering the interaction between developers and users as a network, and uses the quantitative indicators proposed by Freeman [58].

7.2 Analysing the Social Networks constituted by Open Source communities

To adopt Social Network Analysis (SNA) approach data and results obtained during previous study Chap. 5 were organized as follows:

- **nodes** are the mail senders, in other words each community member who posted a message in a discussion thread
- **links** are established between two members participating in the same thread.

In this research, two nodes (community members) are connected if both participated to the same thread at least once.

[1] Every situation where a skilled person helps and supports a less experienced one

If a thread involved several members, the sub-network composed by these members is fully connected.

7.3 Data analysis

The network built was analyzed extracting the three centrality measures previously introduced Chap. 6:*degree*, *betweenness* and *closeness* indexes.

7.3.1 Degree index

The degree is the simplest measure of centrality, defining as most central those actors who have the largest number of ties to other actors in the network [55]. Freeman [58], reviewing previous related work, chose Nieminen's definition for point degree [59]:

$$C_D(m_i) = \sum_j m_{ij} \tag{7.1}$$

where m indicates the community member, and $m_{ij} = 1$ if m_i and m_j are connected, 0 otherwise.

The group degree centralization [58] index is defined as follows:

$$C_D = \frac{\sum_{i=1}^{g} \left[C_D(n^{'}) - C_D(n_i) \right]}{(g-1)(g-2)} \tag{7.2}$$

where $C_D(n^{'})$ is the largest point degree value and g is the number of nodes in the network. This index equals 1 when one actor interacts with all other $g-1$ actors, and they interact only with him; its minimum value is 0, when all degree values are equal (the graph is *regular*).

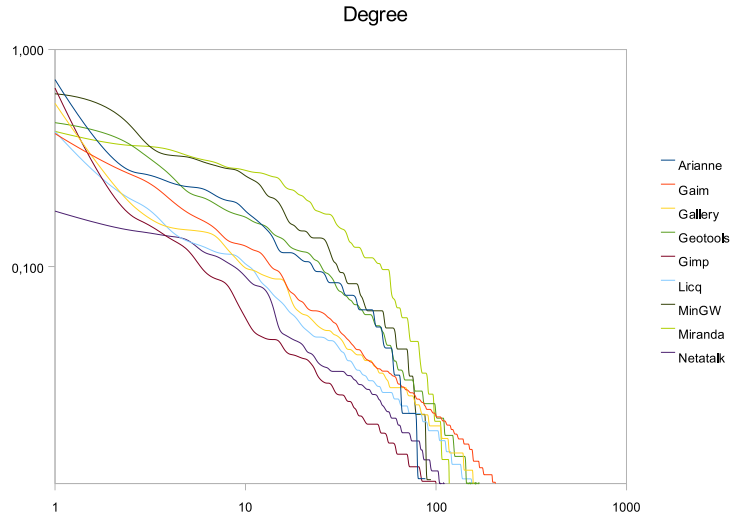

Figure 7.1: Projects' degree distribution

Table 7.1: Mean degree and group degree centralization

Degree			
Project	**Nodes**	**Mean**	**Centralization**
Arianne	95	0.077	0.656
Gaim	1185	0.008	0.402
Gallery	435	0.016	0.551
Geotools	301	0.031	0.431
Gimp-Print	587	0.008	0.658
Licq	572	0.011	0.407
MinGW	97	0.103	0.535
Miranda	156	0.083	0.341
Netatalk	701	0.006	0.174

The degree distribution in log-log scale (Fig. 7.1) shows a similar behavior for all the analyzed projects. Very few central members have a high degree, while the vast majority exhibit quite a small number of direct connections; this is a feature, as noted by Barabasi and Albert [60], common to all scale-free networks. Table 7.1, which lists the number of nodes of each project, mean degree and group degree centralization, shows that some projects have

a low centralization index; these communities (in particular Miranda and Netatalk) do not have a central member with a much higher degree than the other participants, but instead a core group of people (the community *core members*) who are more visible and exchange a larger number of messages. This phenomenon can be also seen in Fig. 7.1, which shows how distribution of the above two projects decreases much more slowly, while Gimp-Print, which has only one prominent member, has the most rapid decay. This explains the substantial difference in degree centralization.

7.3.2 Betweenness index

As Wasserman points out [55], not all actors are connected to the network core, so using the node degrees the position of an actor risks to be evaluated only with respect to the closest ties rather than to the ties as a whole. The betweenness index overcomes this obstacle. Indeed, betweenness is a measure which takes into account, as Freeman suggested, how often a node lies along the shortest path (*geodesic path*) between two other nodes. Under the hypothesis that two geodesics with the same length have the same probability of being chosen, he defined [61] actor betweenness as:

$$C_B(n_i) = \sum_{j<k} \frac{g_{jk}(n_i)}{g_{jk}} \tag{7.3}$$

where g_{jk} is the number of geodesics linking two actors j and k, and $g_k(n_i)$ is the number of geodesics linking those actors and including actor i. In fact, the ratio

$$\frac{g_k(n_i)}{g_{jk}} \tag{7.4}$$

is the probability that the node i falls in the path between the nodes j and k. The group betweenness centralization [58] index is defined as follows:

$$C_B = 2\frac{\sum_{i=1}^{g}\left[C_B(n') - C_B(n_i)\right]}{(g-1)^2(g-2)} \tag{7.5}$$

where $C_B(n')$ is the largest point betweenness value.

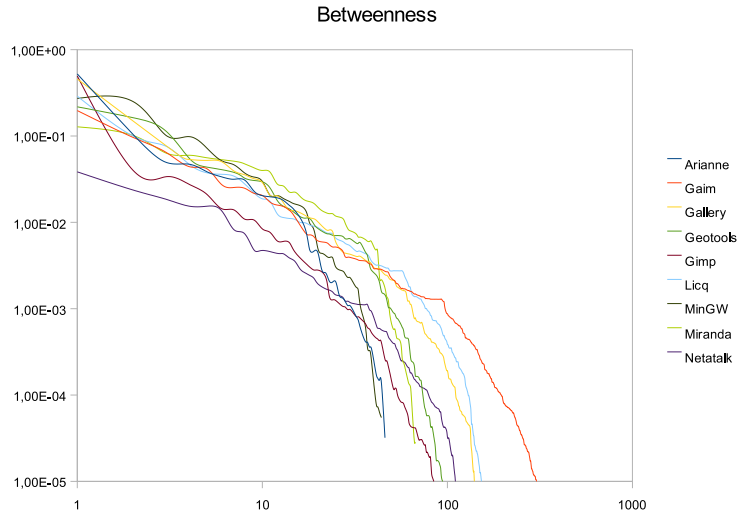

Figure 7.2: Projects' betweenness distribution

Table 7.2: Mean betweenness and group betweenness centralization

Betweenness			
Project	**Nodes**	**Mean**	**Centralization**
Arianne	95	1.11E-2	0.513
Gaim	1185	7.58E-4	0.196
Gallery	435	2.62E-3	0.456
Geotools	301	3.49E-3	0.214
Gimp-Print	587	1.28E-3	0.494
Licq	572	1.77E-3	0.287
MinGW	97	1.25E-2	0.263
Miranda	156	6.88E-3	0.122
Netatalk	701	3.16E-4	0.038

Betweenness, reported in Fig. 7.2, exhibits a behavior similar to degree distribution, that is few members with a high index and uniformity of trend, apart from those projects with low centralization. This is confirmed by the mean betweenness and group betweenness centralization shown in Table 7.2.

7.3.3 Closeness index

The third index examined is closeness; as the name suggests, it measures how close a node is to all the other nodes in the network, being connected with the inverse geodesic distance. The measure chosen by Freeman in his review is Sabidussi's index [62]:

$$C_C(n_i) = \left[\sum_{j=1}^{g} d(n_i, n_j)\right]^{-1} \tag{7.6}$$

where $d(n_i, n_j)$ is the number of lines in the geodesic linking actors i and j, so the sum in the square brackets is the total distance between i and all the other actors.

One problem of this index lies in the fact that isolated nodes (if the graph is disconnected) have infinite distance to the unreachable nodes, thus the index will be zero. For this reason, we only considered non null closeness values.

As far as graph centralization is concerned, Freeman group closeness centralization [58] is:

$$C_C = \frac{\sum_{i=1}^{g}\left[C'_C(n') - C'_C(n_i)\right]}{(g-1)^2(g-2)/2(g-3)} \tag{7.7}$$

where $C'_C(n')$ is the largest point closeness value (the symbol $<'>$ indicates that it is a normalized index, irrespective of network size).

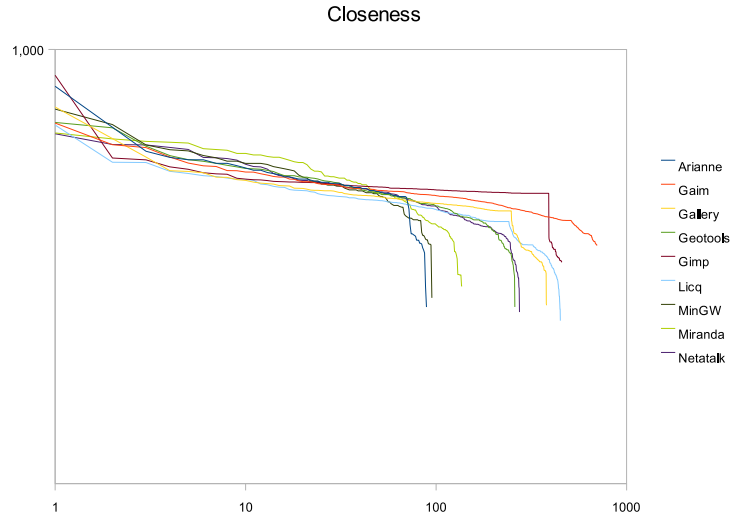

Figure 7.3: Projects' closeness distribution

Table 7.3: Mean closeness and group closeness centralization

Closeness			
Project	**Nodes**	**Mean**	**Centralization**
Arianne	95	0.467	0.716
Gaim	1185	0.400	0.554
Gallery	435	0.409	0.658
Geotools	301	0.423	0.515
Gimp-Print	587	0.455	0.837
Licq	572	0.387	0.567
MinGW	97	0.463	0.540
Miranda	156	0.447	0.383
Netatalk	701	0.417	0.447

The closeness distribution of the projects (Fig. 7.3) and closeness mean and centralization (Table 7.3) show that, unlike the other two measures, closeness values do not differ substantially from one project to another. It was not observed any low centralization, an indication that, as far as closeness is concerned, in general core groups are not present. On the other hand, there is always just one member who can reach the other actors via a much

shorter path.
The narrow range of mean values confirms that, compared to the other indexes, closeness exhibits even less different behaviors across the projects.

7.4 Open Source Communities as Social Networks

By identifying actors with a central position, it was established the existence of elements characterizing the networks corresponding to analyzed communities. These actors, thanks to their particular relevance and visibility, are able to easily maintain contacts with the other members, as it is shown in Fig. 7.4.

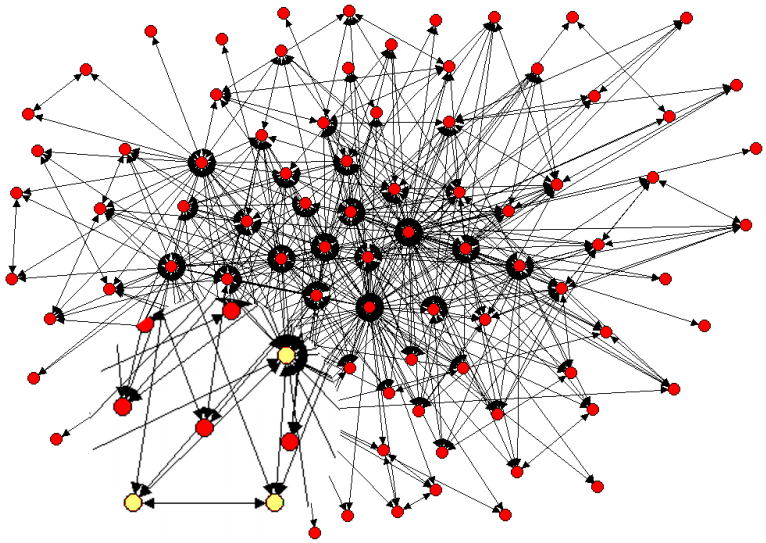

Figure 7.4: An example of communication between developers and users

One of the reasons of their prominence is the fact that, in OS development model, the code is open so it is easy to keep under control the whole development process and its progresses and improvements. This is a commonly adopted practice in Agile methodologies; for example in XP it is known as *collective code ownership* [7].
In practice, in the studied projects, there is one developer, or a few developers, who monitors the mailing list and answers to the questions and the issues raised by other users. A reply starts a thread that may finish there,

or cause other mails and replies by the same or other users. The developers end to be connected to many, or most other users, while these users are connected among them only if they participate to the same thread.
This developers' leadership role is confirmed by what was finding on the SNA indexes. Based on previous studies [63] [64] [61], it is possible to affirm that the degree of a point is an important indicator of communication activity within a network. It is also useful for predicting group performance (for example efficiency, satisfaction, speed). With information about the degree it is also possible quantify the activity conducted by each actor and the subsequent popularity. In fact it is perceived as "*an index of potential communication activity*" [65]. In this sense, degree can be viewed as an indicator of coordination and authority within a group. The degree centralization is highest when there is only one developer, and other users are seldom involved in the thread started by a user. This is the case of Arianne and Gimp-Print projects.
Another aspect to be taken into account is that if two members in a network are not immediate neighbours, then communication will depend on the actors lying between them, so these nodes will exert some sort of control on actors that are not directly connected. Therefore it is possible to view the betweenness index as a measure of actual and potential information flow among members belonging to the same network [55].
The results suggest that the greater the betweenness, the greater the role of hub connecting a large part of the network played by an actor. This is the case of developers who participate in many discussions and provide assistance to numerous users.
Another meaning it is possible to associate with betweenness in analyzed network is that this measure could be an indicator of brokerage and intermediation [65], as *"betweenness of a point is measured by the extent to which the agent can play the part of a 'broker' or gatekeeper"* [66].
A broker is an individual who controls most of the communication flow through the network, so betweenness is a good predictor for identifying leadership. The central member of a network usually emerges as leader and coordinates network activities, helps to solve problems, and facilitates communication [67]. This role of gatekeeper means that the central actor (and in

some cases the core members) could also monitor all the nodes, redistributing information among members, indicating that betweenness is the most useful index for assessing coordination.

In the presented results, the square of degree centralization is highly correlated with betweenness centralization. This suggests that, in a network with a few hubs like those studied, only one of these two indicators can be taken into account.

It has been demonstrated [67] that actors in a central position often have an influence on less connected (weak) nodes too. This behavior can be recognized in every OS community, because information is shared by all members at all levels, and even people occupying a peripheral position in the network can access the same amount of information.

Another research [68] showed that leadership, coordination, connectivity with weak nodes and ease of communication are determined by the same community dynamics. This study suggests that "*there is a correlation between the attributes of social networks, such as density and betweenness centrality, and group productivity measures*" in OS communities. This is confirmed by the fact that the projects there were chosen for our research are some of the most successful and mature in the free software world.

In all examined communities it was found that some members, with high closeness, communicate utilizing short paths, improving, as Freeman affirmed [58] [61], the perception of leadership and group efficiency. In fact, he started from the assumption that the closeness of a point is determined by its independence from the other points in a graph, enabling information to be sent from one actor to another in a network in a short time (that is, with few hops).

Beauchamp arrived at similar conclusions [69], finding this index helpful for describing the network of famous actors: connections via short paths facilitated interrelationships.

Like betweenness, the closeness index also takes into account indirect ties, as observed by Hossain [65], substantiating that the more central is an actor, the more rapidly he can contact the other members. A community leader can quickly reach all the other members even if not directly connected to them. All the studied projects exhibit a high closeness centralization value,

denoting that the communities are strictly tied, irrespectively of the number of hubs.

Conclusion and further works

Conclusion

In latest years Open Source communities have been one of the most studied phenomenon in software development ambit.
There is a huge amount of documents and researches concerning it: surveys, questionnaires, experimental studies...
This is also a multifaced phenomenon concerning different aspects, often related among themselves as, for example, programming language, lines of code, bugs reporting, project maturity.
Two of the most peculiar characteristics of OS communities are *voluntary participation* and *team dislocation*. In fact, members often participate at different software development processes at the same time, self organizing and deciding how and when to work to each project.
At the same time people involved in OSS development are dislocated all over the world: they work, dream, live and eat in different moments of the day. For these important reasons communities need an efficient and well organized system of coordination to better communicate.
There are a lot of very useful generic tools to do that: wiki, forum, instant messaging, chat...
In particular, in recent years OS communities have developed, or customized, appropriate tools to improve the development process: bug tracking system, SVN repository and mailing list. MLs, in fact, are the communicational bonding agent of OS communities: each project has almost one. For this work there were considered developers mailing lists because they are the virtual place where developers and users are used to meet. In fact, the early

researches, at starting phase of this work, underlined the importance of communication both in dis-located and OSS software development teams and the spontaneous appearance on the field of one, or two at least, participant(s) playing *key member* role. A key member connects the most skilled members (developers, bug fixers and managers), often standing for them, with other members, sharing information and giving suggestion. So the research was moved to a larger scale in order to find whether the same behaviour was present in more complicated and organized projects.

A perfect scenario for this study was constituted by SourceForge website, in which there are more than 74.000 OSS projects and all the tools necessary to allow an efficient software development process. There were chosen DMLs of the most active and successful projects and evaluated in two different perspectives based on *empirical metrics* (e-mails exchange, threads started, link among participants) and *quantitative metrics* (obtained thanks to SNA approach).

The novelty of this study consists in matching empirical and quantitative metrics in order to obtain the same result: draw a detailed picture of OSS communicational situation.

In fact, the social network analysis of OS communities makes possible to better describe interactions and communication flow among members of developers MLs. It was found the SNA indexes used to describe the networks suited to individuate the most prominent actors in the communities. They display leadership behaviors and play a major role in team coordination, information management and sharing.

The synthetic indicators of network features, that is the degree, betweeness and closeness centralization can be used to characterize with a few parameters a network, easing to discriminate among various possible structures and flows of control. As it is known by people involved in this research, this is the first time that such analysis is applied to significant collaboration networks using an empirical approach based on quantitative records, so a comparison with studies of the same kind cannot be made.

Further works

This research is carrying on with the inclusion in the analysis of some of the most important Apache Foundation projects. The aim is, by considering larger and more structured communities and comparing them with those analyzed in this work, to discover whether the same communicational dynamics can be found in such different projects.

The final aim of this impressive and articulated research is to compare metrics obtained utilizing SNA approach with process metrics (bug reports, new versions and features releasing, project maturity) in order to compare data related to communication with all the other indexes constituting an OSS project.

In fact, they differ greatly in size, maturity, spread, internal organization, practices, roles... This will be a further step for the understanding and improvement of OS communities and their communicational processes.

This goal is made easy also by feedback of people participating at conference in which this work was presented [56] [57] [70]; it is also hoped for conferences in which it is going to be presented [71] [72]. In fact, during each conference participants showed interest and were very curious about this new and charming studying approach to OS communities, and, surprisingly, almost always a question was about what it was going to be done in the next research step.

This could be interpreted like an empirical affirmation that software engineers need to have more information about communicational processes and their role in OSS communities.

Acknowledgements

Once, in an interview, I read that loving the world means also "*putting back trolley at supermarket*". I think that some people, in the last important eight months, have helped me to put back trolley at supermarket-life. These amazing and wonderful people have done different things as telling me the right word in the right moment, giving me daily the appropriate tools to pursue the happiness, offering me a roof under which stay, making a cake, urging me to work, doing as if I would be a koala bear, believing in me, offering friendship and love... so, rigorously in alphabetical order, I put their names in the right place, here.

Carlo & Matteo, Chicchi-Sabry, Claudia-Kim, DanyCrux, Fede-CicciPucci, Fra & Roby & Vale, Giulio, GioGio & Angelo, Socia-Giordana, Gra, Guido, Lally, Lu, Luc, Marta-Chicca & Edy & Dany, Padrino & Madrina, Pippa & Pippo (my mummy and my daddy),Volf, zio Giuseppe e Tanina, zia Marilena e zio Franco.

Thanks to all :*

Sele

Bibliography

[1] Steiner I.D. *Group process and productivity.* Academic Press, New York, 1972.

[2] Triplett N.D. The dynamogenic factor in pacemaking and competition. *American Journal of Psychology*, 9:507–533, 1961.

[3] Thomas E.J. and Fink C.F. Models of group problem solving. *Journal of Abnormal and Social Psychology*, 63:55–63, 1961.

[4] Bales R.F. and Slater P.E. *Role differentiation in small decision-making groups.* Free Press, 1955.

[5] AgileAlliance. Manifesto for agile software development. http://agilemanifesto.org, 2001.

[6] Watzlawick P., Beavin J.H., and Jackson D.D. *Pragmatics of human communication.* Norton, New York, 1967.

[7] Beck K. *Extreme Programming Explained: Embrace Change.* Addison Wesley, 1999.

[8] McLuhan M. *Understanding Media: The Extensions of Man.* MIT Press, MA, 1964/1994.

[9] Raymond E.S. The cathedral and the bazaar. http://www.firstmonday.org, 1998.

[10] Weber S. *The success of Open Source.* Harvard University Press, MA/UK, 2004.

[11] Torvalds L. Linux history. http://li.org/li/linuxhistory.html, 1999.

[12] Gupta M. and Singh R. An integration-theoretical analysis of cultural and developmental differences in attribution of performance. *Developmental Psychology*, 17:816–825, 1981.

[13] Camplani R., Cau A., Concas G., Mannaro K., Marchesi M., Melis M., Pinna S., Serra N., Setzu A., Turnu I., and Uras S. A comparison between co-located and distributed agile methodologies. In *Sharing Experiences on Agile Methodologies in Open Source Software (OSS 2005)*, pages 9–17, Como, Italy, June 2005.

[14] Camplani R., Cau A., Concas G., Mannaro K., Marchesi M., Melis M., Pinna S., Serra N., Setzu A., Turnu I., and Uras S. Using extreme programming in a distributed team. In *Sharing Experiences on Agile Methodologies in Open Source Software (OSS 2005)*, pages 39–46, Como, Italy, June 2005.

[15] Ye Y. and Kishida K. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering*, 2003.

[16] Lave J. and Wenger E. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, UK, 1991.

[17] Nakakoji K., Yamamoto Y., Nakakoji, Nishinaka K., Kishida K., and Ye Y. Evolution patterns of open source software systems and communities. In *Proceedings of International Workshop on Principles of Software Evolution, IWPSE 2002*, pages 76–85, June 2002.

[18] Schon D.A. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[19] Ragab K., Horikoshi N.Y., Kuriyama H., and Mori K. Autonomous decentralized community communication for information dissemination. *IEEE Internet Computing*, 8(3):29–36, May-Jun 2004.

[20] Jensen C. and Scacchi W. Collaboration, leadership, control, and conflict negotiation and the netbeans.org open source software development

community. In *Proceedings 38th Hawaii International Conference Systems Sciences*, 2005.

[21] Star S. L. The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving. In *Distributed Artificial Intelligence*, pages 37–54, 1990.

[22] Scacchi W. Understanding the requirements for developing open source software. In *Proceedings Software IEE*, pages 24–39, 2002.

[23] Beecham S., Sharp H., Baddoo N., Hall T., and Robinson H. Does the xp environment meet the motivational needs of the software developer? an empirical study. In *Proceedings of Agile 2007*, pages 37–48. IEEE Computer Society Press, 2007.

[24] Whitworth E. and Biddle R. The social nature of agile teams. In *Proceedings of Agile 2007*, pages 26–36. IEEE Computer Society Press, 2007.

[25] Lewin K. Action research and minority problems. *Social Issues*, 2(4):34–46, 1946.

[26] Fowler M. Using an agile software process with offshore development. http://www.martinfowler.com/articles/agileOffshore.html, 2004.

[27] Martin A., Biddle R., and Noble J. When xp met outsourcing. In Eckstein J. and Baumeister H., editors, *Extreme Programming and Agile Processes in Software Engineering*, pages 51–59, 2004.

[28] Braithwaite K. and Joyce T. Xp expended distributed extreme programming. In Eckstein J. and Baumeister H., editors, *Extreme Programming and Agile Processes in Software Engineering*, pages 180–188, 2004.

[29] Poole C. J. Distributed product development using extreme programming. In Eckstein J. and Baumeister H., editors, *Extreme Programming and Agile Processes in Software Engineering*, pages 60–67, 2004.

[30] Beck K. *Extreme Programming Explained: Embrace Change*. Addison Wesley, second edition, 2005.

[31] AgileGroup. Xpswiki. http://www.agilexp.org/xpswiki, 2003.

[32] Stotts D. Virtual teaming: Experiments and experiences with distributed pair programming. In *XP/Agile Universe*, pages 129–141, 2003.

[33] Hanks B. F. Distributed pair programming: An empirical study. In *XP/Agile Universe*, pages 81–91, 2004.

[34] Sobalipse. http://sourceforge.net/projects/sobalipse/, 2003.

[35] Skype. http://www.skype.com/, 2003.

[36] Eclipse. http://www.eclipse.org, 2003.

[37] Subversion. http://subversion.tigris.org/.

[38] Turnu I., Melis M., Cau A., Setzu A., and Concas G. Modeling and simulation of open source development using an agile practice. *Journal of Systems Architecture*, 52(11):610–618, 2006.

[39] McNutt L. and Brennan M. Learning styles and elearning, what is the connection? In *FIE 05, Proceedings 35th Annual conference, USA*, 2005.

[40] Bunse C., Grutzner I., Ochs M., Peper C., and Steinbach Nordmann S. Applying a blended learning strategy for software engineering education. In *Proceedings of the 18th Conference on Software Engineering Education and Training (CSEE & T)*, Ottawa, Canada, 2005.

[41] Beaton.C. Evolution of ethics blended learning. In *Proceedings of the 18th Conference on Software Engineering Education and Training (CSEE & T)*, Ottawa, Canada, 2005.

[42] Moodle. http://moodle.org.

[43] Shih-Wei C. and Chien-Hung L. Learning effectiveness in a web-based virtual learning environment: a learner control perspective. *Journal of Computer Assisted Learning*, 21(1):65–76, 2005.

[44] Infoglue. http://www.infoglue.org.

[45] Compiere. http://www.compiere.org/.

[46] Rogovin A. *Learning by doing: home and school activities for all children.* Abingdon Press, 1998.

[47] Eclipse. http://www.eclipse.org.

[48] Crowston K., Annabi H., Howison J., and Masango C. Effective work practices for floss development: A model and propositions. In *Proceedings of Thirty Eighth Hawai International Conference on System Science, HICSS 38*, Kona, USA, 2005.

[49] Lehmann F. Floss developers as a social formation. http://www.firstmonday.org.

[50] Ghosh R.A. Free/libre/open source software as a learning environment. In *Proceeding of International Symposium on Open Source Software*, Abano Terme, Italy, 2005.

[51] Concas G., Marchesi M., Piras A., Sanna S., Sciola E., and Uras S. Polaris4os: a best practice for training and adoption of floss in sme. In *Towards Open Source Software Adoption, OSS 2006, tOSSad workshop*, pages 89–98, Como, June 2006.

[52] Sourceforge. http://sourceforge.net/.

[53] Crowston K. and Scozzi B. Open source software projects as virtual organizations: competency rallying for software development. In *Proceedings of the First International Conference On Open Source Systems*, 2002.

[54] Crowston K. and Howison J. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[55] Wasserman S. and Faust K. *Social network analysis: methods and applications.* Cambridge University Press, 1994.

[56] Uras S., Concas G., Lisci M., Marchesi M., and Pinna S. Communication flow in open source projects: An analysis of developers' mailing

lists. In Concas G., Damiani E., and Scotto M., editors, *Proceedings of XP2007, Agile Processes in Software Engineering and Extreme Programming*, pages 261–265, Como, June 2006. Springer LNCS.

[57] Concas G., Lisci M., Pinna S., Porruvecchio G., and Uras S. Learning communities in open source projects. In *Proceedings of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2007)*, pages 73–78, Algarve, Portugal, December 2007. IADIS.

[58] Freeman L. C. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1979.

[59] Nieminen J. On centrality in a graph. *Scandinavian Journal of Psychology*, 15(4):322–336, 1974.

[60] Barabasi A. L. and Albert R. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[61] Freeman L. C. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, March 1977.

[62] Sabidussi G. The centrality index of a graph. *Psychometrika*, 31(4):581–603, December 1966.

[63] Bavelas A. and Barret D. An experimental approach to organizational communication. *Personnel*, 27:366–371, 1951.

[64] Leavitt H. J. Some effects of certain communication patterns on group performance. *Journal of Abnormal and Social Psychology*, 46(1):38–50, January 1951.

[65] Hossain L., Wu A., and Chung K. K. S. Actor centrality correlates to project based coordination. In *20th anniversary conference on Computer Supported Cooperative Work*, pages 363–372, New York, November 2006. ACM Press.

[66] Scott J. *Social Network Analysis: A Handbook*. SAGE Publication, London, 2000.

[67] Mullen B., Johnson C., and Salas E. Effect of communication network structure: components of positional centrality. *Social Networks*, 13(2):169–186, 1991.

[68] Kidane Y. H. and Gloor P. A. Correlating temporal patterns of the eclipse open source community with performance and creativity. *Computational and Mathematical Organization Theory*, 13(1):17–27, March 2007.

[69] Beauchamp M. A. An improved index of centrality. *Behavioral Science*, 10:161–163, April 1965.

[70] Concas G., Lisci M., Pinna S., Porruvecchio G., and Uras S. Analysing the social networks constituted by open source communities. In *Proceedings of IECCS 07*, December 2007.

[71] Concas G., Lisci M., Pinna S., Porruvecchio G., and Uras S. Open source communities as social networks: an analysis of some peculiar characteristics. *Unpublished*.

[72] Porruvecchio G., Uras S., Concas G., Marchesi M., Pinna S., and Quaresima R. Social network analysis of communication in open source projects. *Unpublished*.